

Introduction to OSA Programming

This chapter introduces the OS/2 Open Scripting Architecture programming notations and conventions used in this reference. This chapter contains important information and should be read before using this reference.

OSA is composed of the following components:

- The *Object Support Library* (OSL) provides a standardized way for a client application to generate *object specifier records* that identify the objects to be operated on by an OSA event "verbs". They are in the form of structured names, or "noun phrases".

OSA also provides a rich set of *formTypes* for specifying "light weight" objects within the context of a document; for example, "all words whose font is "Courier" and begin with "a"". These functions are described in [Object Support Library Functions](#).
- The *OSA Event Manager* is a set of functions which allows an application to send and receive OSA events. In addition, it supports the registration of callback routines to process OSA events. These functions are described in [OSA Event Manager Functions](#).
- A set of standard scripting component classes, methods, and data structures supports attaching scripts, written in any OSA compatible scripting language, to user interface objects such as words, spread sheets, buttons or menu items. This supports "smart objects" that allow users to tailor an application to their unique requirements. These classes and methods are described in [OSA Classes and Methods](#).
- The *OSA Event Registry: Standard Suites* is a published standard which defines the methods with parameters and classes with properties which an application or OpenDoc part can expose as objects for manipulation by OSA events. It defines the "nouns and verbs" which applications should support to provide a consistent scripting interface.

Since suites standardize or "flatten" the basic scripting interface, applications and parts can work together seamlessly. Scripts written to automate one part can, in general, work with all similar parts.

Other approaches, which allow applications to expose unique interfaces require that all applications support the interfaces of all other applications. They require a scriptor to know the custom interfaces of all parts. They do not allow simple scripts to continue to work correctly even if an underlying part editor is replaced by another.

The OpenDoc environment allows arbitrary content and part editors to be added to and deleted from a document dynamically. The basic scripting interface of these parts must be standardized so that document scripts can operate on a wide variety of embedded parts.

The following chapters provide important information about OSA functions, methods, constants, and data types defined in these components. It provides language-dependent information about the functions that enable the user to call functions in the C programming language and methods in the C++ or SOM programming languages.

Conventions Used in This Book

The following notation and document conventions are used in this book.

Notation Conventions

The following notation conventions are used in this book:

NULL	The term NULL applied to a parameter is used to indicate the presence of a pointer parameter that has no value.
NULLHANDLE	The term NULLHANDLE applied to a parameter is used to indicate the presence of the handle parameter that has no value.
Implicit Pointer	If no entry for a data type "Pxxxxxxx" exists in Data Types , then it is implicitly a pointer to the data type "xxxxxxx". See Implicit Pointer Data Types for more information about implicit pointers.

Documentation Conventions

Throughout this library of documents, the following conventions distinguish the different elements of text:

bold	Action bar and menu choices, names of keys, push buttons, check boxes, radio buttons and field names.
plain text	Function names, structure names, data types names, enumerated types, and constant names.
<i>italics</i>	Parameters, structure fields, titles of documents, and first occurrences of words with special meanings.
CAPITALS	File names and error message text.
monospace	Programming examples and user input at the command line prompt or into an entry field.

Conventions Used in Function Descriptions

The following figure illustrates a sample function:

SampleFunctionName

A description of the function.

```
#define INCL_OSA
#include <os2.h>
```

```
dat_return SampleFunctionName (dat_1 param_1, ..., dat_n param_n)
```

Parameter	param_1 (dat_1) - parameter type	A description of param_1.	param_n (dat_n) - parameter type
	A description of param_n.	Returns	param_return (dat_return) - returns	A description of param_return.	Remarks		
	Information about using the function.	Related Functions	A list of related function described in this book.	Example Code	An		
	example of a code segment that uses the function.						

The following list explains the purpose of each item on a reference page.

Heading

The heading shows the function name. Functions are separated by a rule above each name.

Description

The description follows the heading and explains what the function does. It also identifies options and requirements for calling the function.

Syntax

The syntax describes the C-language calling syntax of the function. This includes any #define and #include directives that must be used when calling the functions.

Parameters

This section lists each parameter passed by the function and provides its data type, parameter type, and a brief description. All parameters must be specified when calling the function.

- All data types are written in uppercase and lowercase to match the header files. A data type of "Pxxxxxxx" implicitly defines a pointer to the data type "xxxxxxx".

The term NULL applied to a parameter indicates the presence of a parameter that has no value.

Refer to [Data Types](#), for a complete list of all data types and their descriptions.

- There are three parameter types:

Input

A value that is specified by the programmer, but is not modified by the

function. If the parameter is a pointer to a value, neither the pointer nor the information it points to changes.

Output

A parameter that receives the function's output. If the value represents a pointer, the pointer remains the same, but the information it points to changes.

Input/Output

A value that is provided as input to the function and whose content is changed upon output. If the parameter is a pointer to a value, the pointer remains the same, but the information it points to changes.

- A brief description is provided with each parameter. Where appropriate, restrictions are also included.

Returns

This section includes a list of possible return codes or values.

Remarks

This section contains additional information about the function, such as:

- guidelines for specifying parameters
- instructions about the order in which to use the function with related functions
- when you might want to use one function in place of another.

Related Functions

This list shows other functions (if any) defined in this document that are related to the function being described.

Example Code

This section provides a programming example of how to use the function in an application program. The example uses valid and realistic values for parameters.

In the online version of this book, you can copy the example to a file. This example can be used when you customize your application.

Conventions Used in Method Descriptions

The following figure illustrates a sample method:

SampleMethodName

A description of the method.

```
#define INCL_OSA  
#include <os2.h>
```

```
dat_return SampleMethodName (dat_1 param_1, ..., dat_n param_n)
```

Parameter **param_1** (dat_1) - parameter type A description of param_1. . . . **param_n** (dat_n) - parameter type
A description of param_n. **Returns** **param_return** (dat_return) - returns A description of param_return. **Remarks**
Information about using the method. **Exception Handling** Information about exception handling. **Override Policy** Information
about whether this method can be overridden. **Related Methods** A list of related method described in this book. **Example Code** An
example of a code segment that uses the method.

The following list explains the purpose of each item on a reference page.

Heading

The heading shows the method name. Methods are separated by a rule above each name.

Description

The description follows the heading and explains what the method does. It also identifies options and requirements for calling the method.

Syntax

The method syntax describes the C++-language calling syntax of the method. This includes any #define and #include directives that must be included when calling these methods.

Parameters

This section lists each parameter passed by the method and provides its data type, parameter type, and a brief description. All parameters must be specified when calling the method.

- All data types are written in uppercase and lowercase to match the header files. A data type of "Pxxxxxxx" implicitly defines a pointer to the data type "xxxxxxx".

The term NULL applied to a parameter indicates the presence of a parameter that has no value.

Refer to [Data Types](#), for a complete list of all data types and their descriptions.

- There are three parameter types:

Input	A value that is specified by the programmer, but is not modified by the method. If the parameter is a pointer to a value, neither the pointer nor the information it points to changes.
Output	A parameter that receives the method's output. If the value represents a pointer, the pointer remains the same, but the information it points to changes.
Input/Output	A value that is provided as input to the method and whose content is changed upon output. If the parameter is a pointer to a value, the pointer remains the same, but the information it points to changes.

- A brief description is provided with each parameter. Where appropriate, restrictions are also included.

These methods have two additional parameters that are not shown. The first parameter is a pointer to the object on which the method is being invoked. This does not need to be specified if you are using C++. The second parameter is a pointer to the environment variable. For example, the syntax of the [ComponentManager::CloseComponent](#) method in the SOM language is as follows:

```
OSErr CloseComponent (ComponentManager *somSelf, Environment *ev, Component *theComponentInstance)
```

If you are using the C++ language, the syntax is as follows:

```
OSErr CloseComponent (Environment *ev, Component *theComponentInstance)
```

Returns

This section includes a list of possible return codes or values.

Remarks

This section contains additional information about the method such as:

- guidelines for specifying parameters
- instructions about the order in which to use the method, with related methods
- when you might want to use one method in place of another.

Exception Handling

This section describes the exception handling for each method. The exception handling section is not shown if there are no exceptions for the method.

Override Policy

This section describes when the method can be overridden. The override policy is not shown unless it is different than the default override policy in which a derived class can override the method and can call the parent's copy of the method from within its own copy.

Related Methods

This list shows other methods (if any) defined in this document that are related to the method being described.

Example Code

This section provides a programming example of how to use the method in an application program. The example uses valid and realistic values for parameters.

In the online version of this book, you can copy the example to a file. This example can be used when you customize your application.

Error Severities

Each of the error conditions falls into one of these areas:

Warning

The function detected a problem but took some remedial action that enabled the function to complete successfully. The return code, in this case, indicates that the function completed successfully.

Error

The function detected a problem for which it could not take any sensible, remedial action. The system has recovered from the problem, and the state of the system, with respect to the application, remains the same as at the time when the function was requested. The system has not executed the function, partially or completely. It has only reported the error.

Severe Error

The function detected a problem from which the system could not reestablish its state, with respect to the application, at the time when that function was requested; that is, the system partially executed the function; therefore, the application needs to perform some corrective activity to restore the system to some known state.

Unrecoverable Error

The function detected some problem from which the system could not reestablish its state, with respect to the application, at the time when that call was issued. It is possible that the application cannot perform some corrective action to restore the system to some known state.

Header Files

All functions and methods require a "#include" statement for the system header file OS2.H:

```
#include <OS2.H>
```

Most functions and methods also require a "#define" statement to select an appropriate (conditional) section of the header file and, hence, the required prototype. Where this is necessary, it is shown at the head of the function definition in the form:

```
#define INCL_name
```

Note: These "#define" statements must precede the "#include <OS2.H>" statement.

There are three common INCL_* names defined for OSA functions:

INCL_OSAOSL	Open Scripting Library functions
INCL_OSAEVENTS	OSA Event Manager function
INCL_OSASUITES	Standard suites

Addressing Elements in Arrays

Constants defining array elements are given values that are zero-based in C; that is, the numbering of the array elements starts at zero, not one.

For example, in the DevQueryCaps function, the sixth element of the *alArray* parameter is CAPS_HEIGHT, which is equated to 5.

Count parameters related to such arrays always mean the actual number of elements available; therefore, again using the DevQueryCaps function as an example, if all elements up to and including CAPS_HEIGHT are provided for, *ICount* could be set to (CAPS_HEIGHT+1).

In functions for which the starting array element can be specified, arrays are always zero-based, and so the C element number constants can

be used directly. For example, to start with the CAPS_HEIGHT element, the *lStart* parameter can be set to CAPS_HEIGHT.

Implicit Pointer Data Types

A data-type name beginning with "P" (for example, PSIZEL) is likely to be a pointer to another data type (in this instance, [SIZEL](#)).

In the data-type summary, [Data Types](#), no explicit "typedefs" are shown for pointers; therefore, if no data-type definition is listed in the summary for a data-type name "Pxxxxxx", it represents a pointer to the data type "xxxxxx", for which a definition should be found in the reference.

The implicit type definition needed for such a pointer "Pxxxxxx" is:

```
typedef xxxxxx *Pxxxxxx;
```

Such definitions are provided in the header files.

Storage Mapping of Data Types

The storage mapping of the data types is dependent on the machine architecture. To be portable, applications must access the data types using the definitions supplied for the environment in which they will execute.

Message Queues

Usually, when an application thread uses a Presentation Manager (PM) function, a message queue must be available for that thread. This means that, before calling the function, WinCreateMsgQueue must be called by the same thread.

It is recommended that you create a message queue for every thread that calls any method, because a PM function might be used by the methods you are calling.

Programming Considerations

This section provides information you need to consider before you begin programming with OSA functions and methods.

Stack Size

Existing 16-bit applications (small and tiny models) must have a 4KB stack available when they enter system calls; otherwise, the stack can overflow into the data area.

C++ Considerations

This section contains several topics you should take into consideration if you are using C++.

C++ Header Files

OS/2 functions that used to take a [PSZ](#) as a parameter and do not modify the contents of the passed string, have been updated in the C++ header files to take a [PCSZZ](#) data type parameter. The use of [PCSZZ](#) allows for better optimization by the compiler and is more semantically compatible with C++. Existing code that calls functions that use [PSZ](#) will continue to work correctly.

Several of the typedefs have been changed in the C++ header files. For example, many items that are `unsigned char` in the C header files are `char` in the C++ header files. For example,

```
typedef unsigned char BYTE;
```

has changed to

```
typedef char BYTE;
```

The existing samples that are included in the OpenDoc for OS/2 Toolkit can be used with either set of the header files.

PCSZZ Data Type

Note: The [PCSZZ](#) data type is defined in the C++ header files included with this product. The use of the "const" keyword is not necessarily specific to C++. Certain C compilers support it as well.

If a function takes, as a parameter, a string that is not changed by the function, the string parameter can be declared as a "const" string, or a [PCSZZ](#). [PCSZZ](#) is defined in the C++ header files as a "const" pointer to a NULL-delimited string. The "const" means that the function will not change the contents of the string.

Declaring the parameter as [PCSZZ](#) informs the C++ compiler that the function will not change the string. Therefore, the compiler simply passes a pointer to the string in the function parameter list. If the parameter is declared as a normal [PSZ](#) (not "const"), the compiler assumes that the function might change the string. Under these circumstances, the compiler will add code to make a copy of the string and then pass a pointer to the copy, rather than passing a pointer to the original string.

A smaller, faster executable is often produced if the data item passed in a parameter list is declared as "const".

If the data item is declared as "const", then it must not be changed by the function.

LINK386

The C++ compiler will provide a dynamic link library to be used by LINK386 when generating error messages. This DLL will convert a compiler-generated mangled name into the function prototype. If the DLL is not present, an error message will be displayed and LINK386 will display the compiler-generated mangled name in error messages.

OSA Functions

This section contains an alphabetic list of functions that can be called within PM applications and OpenDoc parts, written in both C and C++ languages.

The functions are divided into the following subsections:

- [Component Manager Functions](#)

- [Object Support Library Functions](#)
- [OSA Event Manager Functions](#)

Component Manager Functions

This chapter provides information about the Component Manager functions, for both C and Object REXX languages. The following lists the component manager functions in alphabetic order.

- [ODInstallComponent](#) (C)
- [ODInstallComponent](#) (OREXX)
- [ODUninstallComponent](#) (C)
- [ODUninstallComponent](#) (OREXX)

ODInstallComponent (C)

ODInstallComponent (C) - Syntax

This function installs the component in the component registration data base.

```
#define INCL_ODAPI
#include <os2.h>
#include <odcmgr.h>

ComponentRegistryData *pcrd;
APIRET rc; /* Return code. */

rc = ODInstallComponent(pcrd);
```

ODInstallComponent (C) Parameter - pcrd

pcrd ([ComponentRegistryData *](#)) - input
A pointer to the component's registration information.

ODInstallComponent (C) Return Value - rc

rc ([APIRET](#)) - returns
Return code.

noErr

No error.

errCMComponentAlreadyInstalled

errCMBadParm	The specified component is already in the registration data base.
errCMSystemError	One of more of the parameters passed were invalid.
	The Component Manager encountered a system error.

ODInstallComponent (C) - Parameters

pcrd ([ComponentRegistryData *](#)) - input
A pointer to the component's registration information.

rc ([APIRET](#)) - returns
Return code.

noErr	No error.
errCMComponentAlreadyInstalled	The specified component is already in the registration data base.
errCMBadParm	One of more of the parameters passed were invalid.
errCMSystemError	The Component Manager encountered a system error.

ODInstallComponent (C) - Example Code

This example sets up the component registry information and adds the scripting component to the component registration data base.

```
ComponentRegistryData crd;
OS_ERR rc;

/* ***** Set up component registry information. ***** */

/* scripting component */
crd.componentType = kOSAComponentType;

/* subtype code for your component in big-endian format 'MYSC' --> 'CSYM' */
crd.componentSubType = 0x4353594D;

/* manufacturer code 'XYZ' */
crd.componentManufacturer = 0x58595A20;

/* flags indicating supported functions */
crd.componentFlags = 0x000001FE;

/* version of component */
crd.componentVersion = 1;

/* SOM Class name for component */
strcpy(crd.componentClassName, "REXXScriptingComponent");

/* DLL name for component without the '.dll' */
strcpy(crd.componentDLL, "rexxsc");
```

```

/* name of the component */
strcpy(crd.componentName,"Object REXX Scripting Component");

/* component info */
strcpy(crd.componentInfo,"(C) IBM Corporation, 1994");

/* Install the component.*/
rc = ODInstallComponent(&crd);

```

ODInstallComponent (C) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

ODInstallComponent

ODInstallComponent - Example Code

This example sets up the component registration information and adds the scripting component to the component registration data base.

```

call RxFuncAdd 'ODLoadCompMgrFuncs', 'OpenDoc', 'ODLoadCompMgrFuncs'
call ODLoadCompMgrFuncs

cType      = ' aso'                /* big-endian format of 'osa ' */
cSubType   = 'CSYM'                /* big-endian format of 'MYSC' */
cMftr      = 'XYZ '
cFlags     = '0x0000017E'
cVer       = '1'
cDLL       = 'mycomp'
cClass     = 'MyScriptingComponent'
cName      = 'My Own Scripting Component'
cInfo      = 'XYZ Corporation, 1995'

rc = ODInstallComponent( cType, cSubType, cMftr, cFlags, cVer, cDLL, cClass
say 'rc = ' rc

call ODUnloadCompMgrFuncs

```

ODInstallComponent - Purpose

This REXX function installs a component in the component registration data base.

ODInstallComponent Keyword - componentType

componentType

The component type, in big-endian format.

ODInstallComponent Keyword - componentSubType

componentSubType

The component subtype, in big-endian format.

ODInstallComponent Keyword - componentManufacturer

componentManufacturer

The four-character name of the manufacturer.

ODInstallComponent Keyword - componentFlags

componentFlags

The flags indicating supported functions.

ODInstallComponent Keyword - componentVersion

componentVersion

The version level of the component.

ODInstallComponent Keyword - componentDLL

componentDLL

The name of the component's DLL, without the .DLL extension.

ODInstallComponent Keyword - componentClassName

componentClassName
The SOM class name.

ODInstallComponent Keyword - componentName

componentName
The component name.

ODInstallComponent Keyword - componentInfo

componentInfo
Additional component information.

ODInstallComponent - Keywords

componentType
The component type, in big-endian format.

componentSubType
The component subtype, in big-endian format.

componentManufacturer
The four-character name of the manufacturer.

componentFlags
The flags indicating supported functions.

componentVersion
The version level of the component.

componentDLL
The name of the component's DLL, without the .DLL extension.

componentClassName
The SOM class name.

componentName
The component name.

componentInfo
Additional component information.

ODInstallComponent - Syntax


```
ODInstallComponent ( componentType , componentSubType ,  
                    componentManufacturer , componentFlags ,  
                    componentVersion , componentDLL ,  
                    componentClassName , componentName , componentInfo )
```

Examples

ODInstallComponent - Topics

Select an item:

[Purpose](#)
[Syntax](#)
[Keywords](#)
[Example Code](#)
[Glossary](#)

ODUninstallComponent (C)

ODUninstallComponent (C) - Syntax

This function removes a component from the component registration data base.

```
#define INCL_ODAPI  
#include <os2.h>  
#include <odcmgr.h>  
  
OSType    comptype;  
OSType    compSubType;  
APIRET    rc;          /* Return code. */  
  
rc = ODUninstallComponent (comptype, compSubType);
```

ODUninstallComponent (C) Parameter - comptype

comptype ([OSType](#)) - input
The component type; for example, **osa** for the component to uninstall.

ODUninstallComponent (C) Parameter - compSubType

compSubType (**OSType**) - input
The component subtype; for example, **OREX** for the component to install.

ODUninstallComponent (C) Return Value - rc

rc (**APIRET**) - returns
Return code.

noErr	No error.
errCMInvalidComponentID	The specified component ID is not found in the registration data base.
errCMSystemError	The Component Manager encountered a system error.

ODUninstallComponent (C) - Parameters

comptype (**OSType**) - input
The component type; for example, **osa** for the component to uninstall.

compSubType (**OSType**) - input
The component subtype; for example, **OREX** for the component to install.

rc (**APIRET**) - returns
Return code.

noErr	No error.
errCMInvalidComponentID	The specified component ID is not found in the registration data base.
errCMSystemError	The Component Manager encountered a system error.

ODUninstallComponent (C) - Example Code

This example removes a scripting component from the component registration data base whose component type is **osa** and whose subtype is **MYSC**.

```
OSErr                rc;
OSType               compType;
OSType               compSubType;

/* scripting component */
compType             = kOSAComponentType;

/* subtype code for your component in big-endian format 'MYSC' --> 'CSYM' */
compSubType          = 0x4353594D;

/* Uninstall the component. */
```

```
rc = ODUninstallComponent(compType, compSubType);
```

ODUninstallComponent (C) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

ODUninstallComponent

ODUninstallComponent - Example Code

This example removes a scripting component from the component registration data base whose component type is **osa** and whose subtype is **MYSC**.

```
call RxFuncAdd 'ODLoadCompMgrFuncs', 'OpenDoc', 'ODLoadCompMgrFuncs'  
call ODLoadCompMgrFuncs
```

```
cType    = ' aso'           /* big-endian format of 'osa ' */  
cSubType = 'CSYM'           /* big-endian format of 'MYSC' */
```

```
rc = ODUninstallComponent(cType, cSubType)  
say 'rc = ' rc
```

```
call ODUnloadCompMgrFuncs
```

ODUninstallComponent - Purpose

This function removes a component from the registration data base.

ODUninstallComponent Keyword - componentType

componentType
The component type, in big-endian format.

ODUninstallComponent Keyword - componentSubType

componentSubType

The component subtype, in big-endian format.

ODUninstallComponent - Keywords

componentType

The component type, in big-endian format.

componentSubType

The component subtype, in big-endian format.

ODUninstallComponent - Syntax

```
ODUninstallComponent ( componentType , componentSubType )
```

Examples

ODUninstallComponent - Topics

Select an item:

[Purpose](#)

[Syntax](#)

[Keywords](#)

[Example Code](#)

[Glossary](#)

Object Support Library Functions

This chapter provides information about the Object Support Library (OSL) functions. These functions are listed in alphabetic order.

The Object Support Library (OSL) allows clients to generate correct object specifier records. These are "noun phrases" that specify a set of target objects of an event. The OSL is also used by servers to parse or "crack" the object specifier record to identify objects in their context. An object specifier record is passed as the direct parameter of OSA events.

The following lists the OSL functions in alphabetic order.

- [AECallObjectAccessor](#)
- [AECreatCompDescriptor](#)
- [AECreatLogicalDescriptor](#)

- [AECreatObjSpecifier](#)
- [AECreatOffsetDescriptor](#)
- [AECreatRangeDescriptor](#)
- [AEDisposeToken](#)
- [AEGetObjectAccessor](#)
- [AEInstallObjectAccessor](#)
- [AERemoveObjectAccessor](#)
- [AEResolve](#)
- [AESetObjectCallbacks](#)

AECallObjectAccessor

AECallObjectAccessor - Syntax

This function invokes one of your application's object accessor functions.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

DescType      desiredClass;
const AEDesc  *containerToken;
DescType      containerClass;
DescType      keyForm;
AEDesc        *keyData;
AEDesc        *token;
OSErr         rc;          /* Return code. */

rc = AECallObjectAccessor(desiredClass, containerToken,
                          containerClass, keyForm, keyData, token);
```

AECallObjectAccessor Parameter - desiredClass

desiredClass ([DescType](#)) - input
The object class of the desired OSA event objects.

AECallObjectAccessor Parameter - containerToken

containerToken ([const AEDesc *](#)) - input
The token that identifies the container for the desired objects.

AECallObjectAccessor Parameter - containerClass

containerClass ([DescType](#)) - input
The object class of the container for the desired objects.

AECallObjectAccessor Parameter - keyForm

keyForm ([DescType](#)) - input
The key form specified by the object specifier record for the object or objects to be located.

AECallObjectAccessor Parameter - keyData

keyData ([AEDesc *](#)) - input
The key data specified by the object specifier record for the object or objects to be located.

AECallObjectAccessor Parameter - token

token ([AEDesc *](#)) - output
The object accessor function that is invoked returns a token specifying the desired object or objects in this parameter.

AECallObjectAccessor Return Value - rc

rc ([OSErr](#)) - returns
Return code.

In addition to the following result codes, AECallObjectAccessor returns any other result codes returned by the object accessor function that is called.

noErr	No error.
errAEAccessorNotFound	No object accessor is found.

AECallObjectAccessor - Parameters

desiredClass ([DescType](#)) - input
The object class of the desired OSA event objects.

containerToken ([const AEDesc *](#)) - input

The token that identifies the container for the desired objects.

containerClass ([DescType](#)) - input

The object class of the container for the desired objects.

keyForm ([DescType](#)) - input

The key form specified by the object specifier record for the object or objects to be located.

keyData ([AEDesc *](#)) - input

The key data specified by the object specifier record for the object or objects to be located.

token ([AEDesc *](#)) - output

The object accessor function that is invoked returns a token specifying the desired object or objects in this parameter.

rc ([OSError](#)) - returns

Return code.

In addition to the following result codes, AECallObjectAccessor returns any other result codes returned by the object accessor function that is called.

noErr

No error.

errAEAccessorNotFound

No object accessor is found.

AECallObjectAccessor - Remarks

This function is not intended for use by OpenDoc part handlers.

If you want your application to do some of the OSA event object resolution normally performed by the [AEResolve](#) function, you can use this function to invoke an object accessor function. This might be useful, for example, if you have installed an object accessor function using typeWildcard for the [AEInstallObjectAccessor](#) function's *desiredClass* parameter and typeAEList for the *containerType* parameter. To return a list of tokens for a request like "every line that ends in a period," the object accessor function can create an empty list, then call AECallObjectAccessor for each requested element, adding tokens for each element to the list one at a time.

The parameters of AECallObjectAccessor are identical to the parameters of an object accessor function, with one exception: the parameter that specifies the reference constant passed to the object accessor function whenever it is called is added by the OSA Event Manager when it calls the object accessor function. To call an object accessor function whose entry in an object accessor dispatch table specifies typeWildcard as the object class, you must specify typeWildcard as the value of the *desiredClass* parameter.

To call an object accessor function whose entry in an object accessor dispatch table specifies cProperty, you must specify cProperty as the *desiredClass* parameter.

AECallObjectAccessor - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AECreatCompDescriptor

AECreatCompDescriptor - Syntax

This function creates a comparison descriptor record.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

DescType    comparisonOperator;
AEDesc      *operand1;
AEDesc      *operand2;
BOOL        disposeInputs;
AEDesc      *theDescriptor;
OSErr       rc;                                /* Return code. */

rc = AECreatCompDescriptor(comparisonOperator,
                           operand1, operand2, disposeInputs,
                           theDescriptor);
```

AECreatCompDescriptor Parameter - comparisonOperator

comparisonOperator ([DescType](#)) - input

The comparison operator for comparing the descriptor records in the *operand1* and *operand2* parameters. These are the currently defined standard comparison operators:

- kAEGreaterThan**
The value of *operand1* is greater than the value of *operand2* .
- kAEGreaterThanEquals**
The value of *operand1* is greater than or equal to the value of *operand2* .
- kAEEquals**
The value of *operand1* is equal to the value of *operand2* .
- kAELessThan**
The value of *operand1* is less than the value of *operand2* .
- kAELessThanEquals**
The value of *operand1* is less than or equal to the value of *operand2* .
- kAEBeginsWith**
The value of *operand1* begins with the value of *operand2* . (for example, the string "operand" begins with the string "opera").
- kAEEndsWith**
The value of *operand1* ends with the value of *operand2* . (for example, the string "operand" ends with the string "and").
- kAEContains**
The value of *operand1* contains the value of *operand2* . (for example, the string "operand" contains the string "era").

AECreatCompDescriptor Parameter - operand1

operand1 ([AEDesc *](#)) - input

An object specifier record.

AECreatCompDescriptor Parameter - operand2

operand2 ([AEDesc *](#)) - input

A descriptor record (which can be an object specifier record or any other descriptor record) whose value is to be compared to the value of *operand1*.

AECreatCompDescriptor Parameter - disposeInputs

disposeInputs ([BOOL](#)) - input

A flag indicating whether the function should dispose of the descriptor records for the two operands.

TRUE

The function should dispose of the descriptor record.

FALSE

The application should dispose of the descriptor record.

AECreatCompDescriptor Parameter - theDescriptor

theDescriptor ([AEDesc *](#)) - output

The comparison descriptor record that was created.

AECreatCompDescriptor Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

One or more parameters passed in are invalid.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEC coercionFail

Data could not be coerced to the requested OSA event data type.

errAEWrongDataType

A wrong OSA event data type is specified.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AECreatCompDescriptor - Parameters

comparisonOperator ([DescType](#)) - input

The comparison operator for comparing the descriptor records in the *operand1* and *operand2* parameters. These are the currently defined standard comparison operators:

kAEGreaterThan	The value of <i>operand1</i> is greater than the value of <i>operand2</i> .
kAEGreaterThanEquals	The value of <i>operand1</i> is greater than or equal to the value of <i>operand2</i> .
kAEEquals	The value of <i>operand1</i> is equal to the value of <i>operand2</i> .
kAELessThan	The value of <i>operand1</i> is less than the value of <i>operand2</i> .
kAELessThanEquals	The value of <i>operand1</i> is less than or equal to the value of <i>operand2</i> .
kAEBeginsWith	The value of <i>operand1</i> begins with the value of <i>operand2</i> . (for example, the string "operand" begins with the string "opera").
kAEEndsWith	The value of <i>operand1</i> ends with the value of <i>operand2</i> . (for example, the string "operand" ends with the string "and").
kAEContains	The value of <i>operand1</i> contains the value of <i>operand2</i> . (for example, the string "operand" contains the string "era").

operand1 ([AEDesc *](#)) - input

An object specifier record.

operand2 ([AEDesc *](#)) - input

A descriptor record (which can be an object specifier record or any other descriptor record) whose value is to be compared to the value of *operand1* .

disposeInputs ([BOOL](#)) - input

A flag indicating whether the function should dispose of the descriptor records for the two operands.

TRUE	The function should dispose of the descriptor record.
FALSE	The application should dispose of the descriptor record.

theDescriptor ([AEDesc *](#)) - output

The comparison descriptor record that was created.

rc ([OSErr](#)) - returns

Return code.

noErr	No error.
ERROR_INVALID_PARAMETER	One or more parameters passed in are invalid.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEC coercionFail	Data could not be coerced to the requested OSA event data type.
errAEWrongDataType	A wrong OSA event data type is specified.
errAENotAEDesc	

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AECreatCompDescriptor - Remarks

This function creates a comparison descriptor record, which specifies how to compare one or more OSA event objects with either another OSA event object or a descriptor record.

The actual comparison of the two operands is performed by the object-comparison function provided by the client application. The way a comparison operator is interpreted is up to each application.

AECreatCompDescriptor - Example Code

For an example of how to use the AECreatCompDescriptor function to create a comparison descriptor record, see [Specifying a Test](#).

AECreatCompDescriptor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AECreatLogicalDescriptor

AECreatLogicalDescriptor - Syntax

This function creates a logical descriptor record.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

AEDescList    *theLogicalTerms;
DescType      theLogicOperator;
BOOL          disposeInputs;
AEDesc        *theDescriptor;
OSErr         rc;                /* Return code. */

rc = AECreatLogicalDescriptor(theLogicalTerms,
                             theLogicOperator, disposeInputs, theDescriptor);
```

AECreatelogicalDescriptor Parameter - theLogicalTerms

theLogicalTerms ([AEDescList *](#)) - input

A list containing comparison descriptor records, logical records, or both. If the value of the *theLogicOperator* parameter is kAEAND or kAEOR, the list can contain any number of descriptors. If the value of the parameter *theLogicOperator* is kAENOT, logically this list should contain a single descriptor record; however, the function will not return an error if the list contains more than one descriptor record for a logical operator of kAENOT.

AECreatelogicalDescriptor Parameter - theLogicOperator

theLogicOperator ([DescType](#)) - input

A logical operator represented by one of the following constants:

kAEAND	and operator
kAEOR	or operator
kAENOT	not operator

AECreatelogicalDescriptor Parameter - disposeInputs

disposeInputs ([BOOL](#)) - input

A flag indicating whether the function should dispose of the descriptor records in the other parameters.

TRUE	The function should dispose of the descriptor record.
FALSE	The application should dispose of the descriptor record.

AECreatelogicalDescriptor Parameter - theDescriptor

theDescriptor ([AEDesc *](#)) - output

The logical descriptor record that was created.

AECreatelogicalDescriptor Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr	No error.
ERROR_INVALID_PARAMETER	One or more of the parameters passed in are invalid.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the descriptor record.
errAECoercionFail	Data could not be coerced to requested OSA event data type.
errAEWrongDataType	The OSA event data type is wrong.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.

AECreateLogicalDescriptor - Parameters

theLogicalTerms ([AEDescList *](#)) - input

A list containing comparison descriptor records, logical records, or both. If the value of the *theLogicOperator* parameter is kAEAND or kAEOR, the list can contain any number of descriptors. If the value of the parameter *theLogicOperator* is kAENOT, logically this list should contain a single descriptor record; however, the function will not return an error if the list contains more than one descriptor record for a logical operator of kAENOT.

theLogicOperator ([DescType](#)) - input

A logical operator represented by one of the following constants:

kAEAND	and operator
kAEOR	or operator
kAENOT	not operator

disposeInputs ([BOOL](#)) - input

A flag indicating whether the function should dispose of the descriptor records in the other parameters.

TRUE	The function should dispose of the descriptor record.
FALSE	The application should dispose of the descriptor record.

theDescriptor ([AEDesc *](#)) - output

The logical descriptor record that was created.

rc ([OSError](#)) - returns

Return code.

noErr	No error.
ERROR_INVALID_PARAMETER	One or more of the parameters passed in are invalid.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the descriptor record.
errAECoercionFail	Data could not be coerced to requested OSA event data type.
errAEWrongDataType	The OSA event data type is wrong.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.

AECreateLogicalDescriptor - Remarks

This function creates a logical descriptor record, which specifies a logical operator and one or more logical terms for the OSA Event Manager to evaluate.

AECreatelogicalDescriptor - Example Code

For an example of how to use the AECreatelogicalDescriptor function to create a logical descriptor record, see [Specifying a Test](#).

AECreatelogicalDescriptor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AECreatObjSpecifier

AECreatObjSpecifier - Syntax

This function creates an object specifier record.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

DescType    desiredClass;
AEDesc      *theContainer;
DescType    keyForm;
AEDesc      *keyData;
BOOL        disposeInputs;
AEDesc      *objSpecifier;
OSErr       rc;                /* Return code. */

rc = AECreatObjSpecifier(desiredClass, theContainer,
                        keyForm, keyData, disposeInputs, objSpecifier);
```

AECreatObjSpecifier Parameter - desiredClass

desiredClass ([DescType](#)) - input
The object class of the desired OSA event objects.

AECreatObjSpecifier Parameter - theContainer

theContainer ([AEDesc *](#)) - input

A description of the container for the requested object, usually in the form of another object specifier record.

AECreatObjSpecifier Parameter - keyForm

keyForm ([DescType](#)) - input

The key form for the object specifier record.

AECreatObjSpecifier Parameter - keyData

keyData ([AEDesc *](#)) - input

The key data for the object specifier record.

AECreatObjSpecifier Parameter - disposeInputs

disposeInputs ([BOOL](#)) - input

A flag indicating whether the function should dispose of the descriptor records for the other parameters.

TRUE

The function should dispose of the descriptor record.

FALSE

The application should dispose of the descriptor record.

AECreatObjSpecifier Parameter - objSpecifier

objSpecifier ([AEDesc *](#)) - output

The object specifier record that was created.

AECreatObjSpecifier Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_INVALID_PARAMETER
One or more of the passed parameters is invalid.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the descriptor record.

errAECocersionFail
Data could not be coerced to the requested OSA event data type.

errAEWrongDataType
The OSA event data type is wrong.

errAENotAEDesc
The descriptor record is invalid.

errAEBadListItem
An operation involving a list item failed.

AECreatObjSpecifier - Parameters

desiredClass ([DescType](#)) - input
The object class of the desired OSA event objects.

theContainer ([AEDesc *](#)) - input
A description of the container for the requested object, usually in the form of another object specifier record.

keyForm ([DescType](#)) - input
The key form for the object specifier record.

keyData ([AEDesc *](#)) - input
The key data for the object specifier record.

disposeInputs ([BOOL](#)) - input
A flag indicating whether the function should dispose of the descriptor records for the other parameters.

TRUE
The function should dispose of the descriptor record.

FALSE
The application should dispose of the descriptor record.

objSpecifier ([AEDesc *](#)) - output
The object specifier record that was created.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_INVALID_PARAMETER
One or more of the passed parameters is invalid.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the descriptor record.

errAECocersionFail
Data could not be coerced to the requested OSA event data type.

errAEWrongDataType
The OSA event data type is wrong.

errAENotAEDesc
The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AECreatObjSpecifier - Remarks

This function assembles an object specifier record from the specified constants and other descriptor records.

AECreatObjSpecifier - Example Code

For information about how to assemble the components of an object specifier record with the AECreatObjSpecifier function, see [Creating Object Specifier Records](#).

AECreatObjSpecifier - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AECreatOffsetDescriptor

AECreatOffsetDescriptor - Syntax

This function creates an offset descriptor record.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

long      theOffset;
AEDesc    *theDescriptor;
OSErr     rc;          /* Return code. */

rc = AECreatOffsetDescriptor(theOffset, theDescriptor);
```

AECreatOffsetDescriptor Parameter - theOffset

theOffset (long) - input

A positive integer that specifies the offset from the beginning of the container (the first element has an offset of 1), or a negative integer that specifies the offset from the end (the last element has an offset of -1).

AECreatOffsetDescriptor Parameter - theDescriptor

theDescriptor (AEDesc *) - output

The offset descriptor record that was created.

AECreatOffsetDescriptor Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the descriptor record.

errAENotAEDesc

The descriptor record is invalid.

AECreatOffsetDescriptor - Parameters

theOffset (long) - input

A positive integer that specifies the offset from the beginning of the container (the first element has an offset of 1), or a negative integer that specifies the offset from the end (the last element has an offset of -1).

theDescriptor (AEDesc *) - output

The offset descriptor record that was created.

rc (OSErr) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the descriptor record.

errAENotAEDesc

The descriptor record is invalid.

AECreatOffsetDescriptor - Remarks

This function creates an offset descriptor record that specifies the position of an element in relation to the beginning or end of its container.

AECreatOffsetDescriptor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AECreatRangeDescriptor

AECreatRangeDescriptor - Syntax

This function creates a range descriptor record.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

AEDesc      *rangeStart;
AEDesc      *rangeStop;
BOOL        disposeInputs;
AEDesc      *theDescriptor;
OSErr       rc;          /* Return code. */

rc = AECreatRangeDescriptor(rangeStart, rangeStop,
                           disposeInputs, theDescriptor);
```

AECreatRangeDescriptor Parameter - rangeStart

rangeStart ([AEDesc *](#)) - input

An object specifier record that identifies the first OSA event object in the range.

AECreatRangeDescriptor Parameter - rangeStop

rangeStop ([AEDesc *](#)) - input

An object specifier record that identifies the last OSA event object in the range.

AECreatRangeDescriptor Parameter - disposeInputs

disposeInputs (BOOL) - input

A flag indicating whether the function should dispose of the descriptor records for the *rangeStart* and *rangeStop* parameters.

TRUE

The function should dispose of the descriptor records.

FALSE

The application should dispose of the descriptor records.

AECreatRangeDescriptor Parameter - theDescriptor

theDescriptor (AEDesc *) - output

The range descriptor record that was created.

AECreatRangeDescriptor Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

Error in parameter list.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the descriptor record.

errAEC coercionFail

Data could not be coerced to the requested OSA event data type.

errAEWrongDataType

A wrong OSA event data type is specified.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AECreatRangeDescriptor - Parameters

rangeStart (AEDesc *) - input

An object specifier record that identifies the first OSA event object in the range.

rangeStop (AEDesc *) - input

An object specifier record that identifies the last OSA event object in the range.

disposeInputs (BOOL) - input

A flag indicating whether the function should dispose of the descriptor records for the *rangeStart* and *rangeStop* parameters.

TRUE

FALSE The function should dispose of the descriptor records.
The application should dispose of the descriptor records.

theDescriptor ([AEDesc](#) *) - output
The range descriptor record that was created.

rc ([OSError](#)) - returns
Return code.

noErr
No error.

ERROR_INVALID_PARAMETER
Error in parameter list.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the descriptor record.

errAEC coercionFail
Data could not be coerced to the requested OSA event data type.

errAEWrongDataType
A wrong OSA event data type is specified.

errAENotAEDesc
The descriptor record is invalid.

errAEBadListItem
An operation involving a list item failed.

AECreatRangeDescriptor - Remarks

This function creates a range descriptor record, which specifies a series of consecutive elements in the same container. Although the *rangeStart* and *rangeStop* parameters can be any object specifier records-including object specifier records that specify more than one OSA event object-most applications expect these parameters to specify single OSA event objects.

AECreatRangeDescriptor - Example Code

For an example of how to use the AECreatRangeDescriptor function to create a range descriptor record, see [Specifying a Range](#).

AECreatRangeDescriptor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEDisposeToken

AEDisposeToken - Syntax

This function deallocates the memory used by a token.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

AEDesc      *theToken;
OSErr       rc;          /* Return code. */

rc = AEDisposeToken(theToken);
```

AEDisposeToken Parameter - theToken

theToken ([AEDesc](#) *) - input
The token to be disposed of.

AEDisposeToken Return Value - rc

rc ([OSErr](#)) - returns
Return code.

In addition to the following result codes, this function also returns result codes returned by the token disposal function that disposed of the token.

noErr	No error.
ERROR_INVALID_PARAMETER	One or more of the parameters passed in are invalid.
errAENotASpecialFunction	The keyword is not valid for a special function.

AEDisposeToken - Parameters

theToken ([AEDesc](#) *) - input
The token to be disposed of.

rc ([OSErr](#)) - returns
Return code.

In addition to the following result codes, this function also returns result codes returned by the token disposal function that disposed of the token.

noErr	No error.
ERROR_INVALID_PARAMETER	One or more of the parameters passed in are invalid.

errAENotASpecialFunction
The keyword is not valid for a special function.

AEDisposeToken - Remarks

When your application calls this function, the OSA Event Manager first calls your application's token disposal function, if you have provided one. If you have not provided a token disposal function, or if your application's token disposal function returns errAEventNotHandled as the function result, the OSA Event Manager calls the system token disposal function if one is available. If there is no system token disposal function or the function returns errAEventNotHandled as the function result, the OSA Event Manager calls AEDisposeToken to dispose of the token.

For information about writing a token disposal function, see [MyDisposeToken](#).

AEDisposeToken - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEGetObjectAccessor

AEGetObjectAccessor - Syntax

This function returns a pointer to an object accessor function and the value of its reference constant.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

DescType      desiredClass;
DescType      containerType;
OSLAccessorUPP *accessor;
long          *accessorRefcon;
BOOL          isSysHandler;
OSErr         rc;          /* Return code. */

rc = AEGetObjectAccessor(desiredClass, containerType,
                        accessor, accessorRefcon, isSysHandler);
```

AEGetObjectAccessor Parameter - desiredClass

desiredClass (DescType) - input

The object class of the OSA event objects located by the requested object accessor function. This parameter can also contain the constant typeWildCard or the constant cProperty.

AEGetObjectAccessor Parameter - containerType

containerType (DescType) - input

The descriptor type of the token that identifies the container for the objects located by the requested object accessor function. This parameter can also contain the constant typeWildCard.

AEGetObjectAccessor Parameter - accessor

accessor (OSLAccessorUPP *) - output

A pointer to the requested object accessor function is returned in this parameter.

AEGetObjectAccessor Parameter - accessorRefcon

accessorRefcon (long *) - output

The reference constant from the object accessor dispatch table entry for the specified object accessor function is returned in this parameter.

AEGetObjectAccessor Parameter - isSysHandler

isSysHandler (BOOL) - input

A value that specifies the object accessor table from which to get the object accessor function and its reference constant.

TRUE

The function is obtained from the system object accessor dispatch table.

FALSE

The function is obtained from the application's object accessor dispatch table.

AEGetObjectAccessor Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

errAEAccessorNotFound

There is no object accessor function for the specified object class and container type.

errAESysHandlerNotLoaded
The system handler could not be loaded.

AEGetObjectAccessor - Parameters

desiredClass (DescType) - input

The object class of the OSA event objects located by the requested object accessor function. This parameter can also contain the constant typeWildcard or the constant cProperty.

containerType (DescType) - input

The descriptor type of the token that identifies the container for the objects located by the requested object accessor function. This parameter can also contain the constant typeWildcard.

accessor (OSLAccessorUPP *) - output

A pointer to the requested object accessor function is returned in this parameter.

accessorRefcon (long *) - output

The reference constant from the object accessor dispatch table entry for the specified object accessor function is returned in this parameter.

isSysHandler (BOOL) - input

A value that specifies the object accessor table from which to get the object accessor function and its reference constant.

TRUE

The function is obtained from the system object accessor dispatch table.

FALSE

The function is obtained from the application's object accessor dispatch table.

rc (OSErr) - returns

Return code.

noErr

No error.

errAEAccessorNotFound

There is no object accessor function for the specified object class and container type.

errAESysHandlerNotLoaded

The system handler could not be loaded.

AEGetObjectAccessor - Remarks

This function is not intended for use by OpenDoc part handlers.

This function returns a pointer to the object accessor function installed for the object class specified in the *desiredClass* parameter and the descriptor type specified in the *containerType* parameter. It also returns the reference constant associated with the specified function. You must supply a value in the *isSysHandler* parameter that specifies from which object accessor dispatch table you want to get the function.

Calling this function does not remove the object accessor function from an object accessor dispatch table.

To get an object accessor function whose entry in an object accessor dispatch table specifies typeWildcard as the object class, you must specify typeWildcard as the value of the *desiredClass* parameter. Similarly, to get an object accessor function whose entry in an object accessor dispatch table specifies typeWildcard as the descriptor type of the token used to specify the container, you must specify typeWildcard as the value of the *containerType* parameter.

To get an object accessor function whose entry in an object accessor dispatch table specifies cProperty (a constant used to specify a property of any object class), you must specify cProperty as the *containerType* parameter.

AEGetObjectAccessor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEInstallObjectAccessor (OS/2)

AEInstallObjectAccessor (OS/2) - Syntax

This function adds an entry for an object accessor function to the application's object accessor dispatch table.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

DescType      desiredClass;
DescType      containerType;
OSLAccessorUPP theAccessor;
long          accessorRefcon;
BOOL          isSysHandler;
OSErr         rc;          /* Return code. */

rc = AEInstallObjectAccessor(desiredClass,
                             containerType, theAccessor, accessorRefcon,
                             isSysHandler);
```

AEInstallObjectAccessor (OS/2) Parameter - desiredClass

desiredClass ([DescType](#)) - input

Object class of the OSA event objects to be located by the object accessor function for this table entry.

AEInstallObjectAccessor (OS/2) Parameter - containerType

containerType ([DescType](#)) - input

The descriptor type of the token used to specify the container for the desired objects. The object accessor function finds objects in containers specified by tokens of this type.

AEInstallObjectAccessor (OS/2) Parameter - theAccessor

theAccessor ([OSLAccessorUPP](#)) - input

Pointer to the object accessor function for this table entry. The *theAccessor* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of an accessor entry point when a system accessor is being installed; therefore, a cast must be used to pass this parameter.

AEInstallObjectAccessor (OS/2) Parameter - accessorRefcon

accessorRefcon (long) - input

A reference constant passed by the OSA Event Manager to the object accessor function whenever the function is called. If the object accessor function does not use a reference constant, this parameter should be set to 0.

To change the value of the reference constant, the AEInstallObjectAccessor method must be called again.

AEInstallObjectAccessor (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to add the handler to the system object access dispatch table.

TRUE

The OSA Event Manager adds the handler to the system object access dispatch table and makes it available to all applications.

FALSE

The OSA Event Manager adds the handler to the application's object accessor dispatch table. When searching for object accessor functions, the OSA Event Manager searches the application's object accessor dispatch table first; it searches the system object accessor dispatch table only if the necessary function is not found in your application's object accessor dispatch table.

AEInstallObjectAccessor (OS/2) Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The handler pointer is NULL or odd, or an invalid module or process name is specified for the system handler.

AEInstallObjectAccessor (OS/2) - Parameters

desiredClass ([DescType](#)) - input

Object class of the OSA event objects to be located by the object accessor function for this table entry.

containerType ([DescType](#)) - input

The **descriptor type** of the token used to specify the container for the desired objects. The object accessor function finds objects in containers specified by tokens of this type.

theAccessor (OSLAccessorUPP) - input

Pointer to the object accessor function for this table entry. The *theAccessor* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of an accessor entry point when a system accessor is being installed; therefore, a cast must be used to pass this parameter.

accessorRefcon (long) - input

A reference constant passed by the OSA Event Manager to the object accessor function whenever the function is called. If the object accessor function does not use a reference constant, this parameter should be set to 0.

To change the value of the reference constant, the `AEInstallObjectAccessor` method must be called again.

isSysHandler (BOOL) - input

A flag indicating whether to add the handler to the system object access dispatch table.

TRUE

The OSA Event Manager adds the handler to the system object access dispatch table and makes it available to all applications.

FALSE

The OSA Event Manager adds the handler to the application's object accessor dispatch table. When searching for object accessor functions, the OSA Event Manager searches the application's object accessor dispatch table first; it searches the system object accessor dispatch table only if the necessary function is not found in your application's object accessor dispatch table.

rc (OSError) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The handler pointer is NULL or odd, or an invalid module or process name is specified for the system handler.

AEInstallObjectAccessor (OS/2) - Remarks

This function is not intended for use by OpenDoc part handlers.

AEInstallObjectAccessor (OS/2) - Example Code

This example shows how to install object accessors in the application table and in the system table. The object accessor installed in the application table finds columns in container tokens of type `typeMyTableToken`. The object accessor installed in the system table finds rows in containers of type `typeMytableToken`.

```
#define INCL_OSAOSL
```

```
#define INCL_OSAAPI
```

```
#include <os2.h>
```

```
/* Application-defined token descriptor type */
```

```
#define typeMyTableToken 0x546C6274 /* "tblT" */
```

```
/* Prototype for object accessor to be installed in application table */
```

[illegible]

```

                                AEDesc *theToken,
                                long *theRefcon);

OSErr myErr;
AESystemHandler sysObjectAccessor; /* system handler structure
char procedureName[] = "MyRowObjectAccessor"; /* name of object accessor */
char moduleName[] = "mymodule.dll"; /* module where the accessor
/* function resides */

sysObjectAccessor.procedureName = procedureName;
sysObjectAccessor.moduleName = moduleName;

/* Install in application table */
myErr = AEInstallObjectAccessor( cColumn, /* desired class */
                                typeMyTableToken, /* container token type */
                                (OSLAccessorUPP)MyColumnObjectAccessor,
                                0, /* reference constant */
                                FALSE); /* install in application
/* table */

/* Install in system table */
myErr = AEInstallObjectAccessor( cRow, /* desired class */
                                typeMyTableToken, /* container token type */
                                (OSLAccessorUPP)&sysObjectAccessor,
                                0, /* reference constant */
                                TRUE); /* install in system table

```

AEInstallObjectAccessor (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AERemoveObjectAccessor (OS/2)

AERemoveObjectAccessor (OS/2) - Syntax

This function removes an object accessor function from the application's object accessor dispatch table.

```
#define INCL_OSAOSL
```

```

#define INCL_OSAAPI
#include <os2.h>

DescType      desiredClass;
DescType      containerType;
OSLAccessorUPP theAccessor;
BOOL          isSysHandler;
OSErr         rc;          /* Return code. */

rc = AERemoveObjectAccessor(desiredClass,
                           containerType, theAccessor, isSysHandler);

```

AERemoveObjectAccessor (OS/2) Parameter - desiredClass

desiredClass ([DescType](#)) - input

Object class of the OSA event objects located by the object accessor function.

To remove an object accessor function whose entry in an object accessor dispatch table specifies typeWildCard as the object class, you must specify typeWildCard as the value of the *desiredClass* parameter. To remove an object accessor function whose entry in an object accessor dispatch table specifies cProperty, the *desiredClass* parameter must be specified as cProperty. The cProperty constant is used to specify a property of any object class.

AERemoveObjectAccessor (OS/2) Parameter - containerType

containerType ([DescType](#)) - input

Descriptor type of the token that identifies the container for the objects located by the object accessor function.

To remove an object accessor function whose entry in an object accessor dispatch table specifies typeWildCard as the descriptor type of the token used to specify the container for the desired object, the *desiredClass* parameter must be specified as typeWildCard.

AERemoveObjectAccessor (OS/2) Parameter - theAccessor

theAccessor ([OSLAccessorUPP](#)) - input

Pointer to the object accessor function to be removed. The *theAccessor* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of an accessor entry point when a system accessor is being installed; therefore, a cast must be used to pass this parameter.

Although the *desiredClass* and *containerType* parameters are sufficient to identify the function to be removed, this parameter guarantees the correct function is removed. If this parameter does not contain a pointer to the object accessor function, its value should be NULL.

AERemoveObjectAccessor (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system object accessor dispatch table.

TRUE

The OSA Event Manager removes the handler from the system object accessor dispatch table.

FALSE

The OSA event removes the handler from the application's object accessor dispatch table.

AERemoveObjectAccessor (OS/2) Return Value - rc

rc ([OSError](#)) - returns
Return code.

noErr

No error.

errAEAccessorNotFound

There is no object accessor function for the specified object class and container type.

AERemoveObjectAccessor (OS/2) - Parameters

desiredClass ([DescType](#)) - input
Object class of the OSA event objects located by the object accessor function.

To remove an object accessor function whose entry in an object accessor dispatch table specifies typeWildCard as the object class, you must specify typeWildCard as the value of the *desiredClass* parameter. To remove an object accessor function whose entry in an object accessor dispatch table specifies cProperty, the *desiredClass* parameter must be specified as cProperty. The cProperty constant is used to specify a property of any object class.

containerType ([DescType](#)) - input
Descriptor type of the token that identifies the container for the objects located by the object accessor function.

To remove an object accessor function whose entry in an object accessor dispatch table specifies typeWildCard as the descriptor type of the token used to specify the container for the desired object, the *desiredClass* parameter must be specified as typeWildCard.

theAccessor ([OSLAccessorUPP](#)) - input
Pointer to the object accessor function to be removed. The *theAccessor* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of an accessor entry point when a system accessor is being installed; therefore, a cast must be used to pass this parameter.

Although the *desiredClass* and *containerType* parameters are sufficient to identify the function to be removed, this parameter guarantees the correct function is removed. If this parameter does not contain a pointer to the object accessor function, its value should be NULL.

isSysHandler ([BOOL](#)) - input
A flag indicating whether to remove the handler from the system object accessor dispatch table.

TRUE

The OSA Event Manager removes the handler from the system object accessor dispatch table.

FALSE

The OSA event removes the handler from the application's object accessor dispatch table.

rc ([OSError](#)) - returns
Return code.

noErr

No error.

errAEAccessorNotFound

There is no object accessor function for the specified object class and container type.

AERemoveObjectAccessor (OS/2) - Remarks

This function is not intended for use by OpenDoc part handlers.

AERemoveObjectAccessor (OS/2) - Example Code

This example shows how to remove object accessors from the application and system tables. The object accessor removed from the application table is one that finds columns in container tokens of type typeMyTableToken. The object accessor removed from the system table is one that finds rows in containers of type typeMyTableToken.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

/* Application-defined token descriptor type */
#define typeMyTableToken 0x546C6274 /* "tblT" */

/* Prototype for object accessor to be removed from application table */
OSErr APIENTRY MyColumnObjectAccessor (DescType desiredClass,
                                       const AEDesc *containerToken,
                                       DescType containerClass,
                                       DescType keyForm,
                                       const AEDesc *keyData,
                                       AEDesc *theToken,
                                       long *theRefcon);

OSErr myErr;
AESystemHandler sysObjectAccessor; /* system handler structure */
char procedureName[] = "MyRowObjectAccessor"; /* name of object accessor */
char moduleName[] = "mymodule.dll"; /* module where the accessor */
/* function resides */

sysObjectAccessor.procedureName = procedureName;
sysObjectAccessor.moduleName = moduleName;

/* Remove from application table */
myErr = AERemoveObjectAccessor(cColumn, /* desired class */
                              typeMyTableToken, /* container token type */
                              (OSLAccessorUPP)MyColumnObjectAccessor,
                              FALSE); /* remove from application
                                      table */

/* Remove from system table */
myErr = AERemoveObjectAccessor(cRow, /* desired class */
                              typeMyTableToken, /* container token type */
                              (OSLAccessorUPP)&sysObjectAccessor,
                              TRUE); /* remove from system table
```

AERemoveObjectAccessor (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEResolve

AEResolve - Syntax

This function resolves an object specifier record in an OSA event parameter.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

const AEDesc      *objectSpecifier;
short             callbackFlags;
AEDesc            *theToken;
OSErr             rc;                /* Return code. */

rc = AEResolve(objectSpecifier, callbackFlags,
               theToken);
```

AEResolve Parameter - objectSpecifier

objectSpecifier ([const AEDesc *](#)) - input
The object specifier record to be resolved.

AEResolve Parameter - callbackFlags

callbackFlags (short) - input

A value that determines what additional assistance, if any, your application can give the OSA Event Manager when it parses the object specifier record. The value is specified by adding the following constants, as appropriate:

kAEIDoMinimum	Supports minimum callbacks only
kAEIDoWhose	Supports formWhose
kAEIDoMarking	Provides marking functions

AEResolve Parameter - theToken

theToken ([AEDesc *](#)) - output

A token that identifies the OSA event objects specified by the *objectSpecifier* parameter. Your object accessor functions may need to create many tokens to resolve a single object specifier record; this parameter contains only the final token that identifies the requested OSA event object. If an error occurs, a null descriptor record is returned.

AEResolve Return Value - rc

rc ([OSErr](#)) - returns

Return code.

In addition to the result codes listed here, this function also returns any result code returned by one of your application's object accessor functions or object callback functions. For example, an object accessor function can return `errAENoSuchObject` when it cannot find an OSA event object, or it can return more specific result codes.

If any object accessor function or object callback function returns a result code other than `noErr` or `errAEEventNotHandled`, this function immediately disposes of any existing tokens and returns. The result code it returns in this case is the result code returned by the object accessor function or the object callback function.

`noErr`

No error.

`ERROR_INVALID_PARAMETER`

One or more of the parameters passed in are invalid.

`errAEHandlerNotFound`

The necessary object callback function is not found (this result is returned only for object callback functions; `errAEAccessorNotFound` is returned when an object accessor function is not found).

`errAEImpossibleRange`

The range is not valid because it is impossible for a range to include the first and last objects that were specified; an example is a range in which the offset of the first object is greater than the offset of the last object.

`errAEWrongNumberArgs`

The number of operands provided for the `kAENOT` logical operator is not 1.

`errAEAccessorNotFound`

There is no object accessor function for the specified object class and token descriptor type.

`errAENoSuchLogical`

The logical operator in a logical descriptor record is not `kAEAND`, `kAEOR`, or `kAENOT`.

`errAEBadTestKey`

The descriptor record in a test key is neither a comparison descriptor record nor a logical descriptor record

`errAENotAnObjectSpec`

The *objectSpecifier* parameter of `AEResolve` is not an object specifier record.

`errAENegativeCount`

An object-counting function returned a negative result.

`errAEEmptyListContainer`

The container for an OSA event object is specified by an empty list.

AEResolve - Parameters

objectSpecifier (*const AEDesc **) - input
The object specifier record to be resolved.

callbackFlags (short) - input
A value that determines what additional assistance, if any, your application can give the OSA Event Manager when it parses the object specifier record. The value is specified by adding the following constants, as appropriate:

kAEIDoMinimum	Supports minimum callbacks only
kAEIDoWhose	Supports formWhose
kAEIDoMarking	Provides marking functions

theToken (*AEDesc **) - output
A token that identifies the OSA event objects specified by the *objectSpecifier* parameter. Your object accessor functions may need to create many tokens to resolve a single object specifier record; this parameter contains only the final token that identifies the requested OSA event object. If an error occurs, a null descriptor record is returned.

rc (*OSErr*) - returns
Return code.

In addition to the result codes listed here, this function also returns any result code returned by one of your application's object accessor functions or object callback functions. For example, an object accessor function can return `errAENoSuchObject` when it cannot find an OSA event object, or it can return more specific result codes.

If any object accessor function or object callback function returns a result code other than `noErr` or `errAEEventNotHandled`, this function immediately disposes of any existing tokens and returns. The result code it returns in this case is the result code returned by the object accessor function or the object callback function.

`noErr`
No error.

`ERROR_INVALID_PARAMETER`
One or more of the parameters passed in are invalid.

`errAEHandlerNotFound`
The necessary object callback function is not found (this result is returned only for object callback functions; `errAEAccessorNotFound` is returned when an object accessor function is not found).

`errAEImpossibleRange`
The range is not valid because it is impossible for a range to include the first and last objects that were specified; an example is a range in which the offset of the first object is greater than the offset of the last object.

`errAEWrongNumberArgs`
The number of operands provided for the `kAENOT` logical operator is not 1.

`errAEAccessorNotFound`
There is no object accessor function for the specified object class and token descriptor type.

`errAENoSuchLogical`
The logical operator in a logical descriptor record is not `kAEAND`, `kAEOR`, or `kAENOT`.

`errAEBadTestKey`
The descriptor record in a test key is neither a comparison descriptor record nor a logical descriptor record

`errAENotAnObjectSpec`
The *objectSpecifier* parameter of `AEResolve` is not an object specifier record.

`errAENegativeCount`
An object-counting function returned a negative result.

`errAEEmptyListContainer`
The container for an OSA event object is specified by an empty list.

AEResolve - Remarks

This function resolves the object specifier record passed in the *objectSpecifier* parameter with the help of the object accessor functions and

object callback functions provided by your application.

For an overview of how the AEResolve function works with object accessor functions, see [Resolving Object Specifier Records](#).

AEResolve - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AESetObjectCallbacks

AESetObjectCallbacks - Syntax

This function specifies the object callback functions to be called for your application.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

OSLCompareUPP      myCompareProc;
OSLCountUPP        myCountProc;
OSLDisposeTokenUPP myDisposeTokenProc;
OSLGetMarkTokenUPP myGetMarkTokenProc;
OSLMarkUPP         myMarkProc;
OSLAdjustMarksUPP  myAdjustMarksProc;
OSLGetErrDescUPP   myGetErrDescProc;
OSErr              rc;                                /* Return code. */

rc = AESetObjectCallbacks(myCompareProc, myCountProc,
                          myDisposeTokenProc, myGetMarkTokenProc,
                          myMarkProc, myAdjustMarksProc, myGetErrDescProc);
```

AESetObjectCallbacks Parameter - myCompareProc

myCompareProc ([OSLCompareUPP](#)) - input

Either a pointer to the object-comparison function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myCountProc

myCountProc ([OSLCountUPP](#)) - input

Either a pointer to the object-counting function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myDisposeTokenProc

myDisposeTokenProc ([OSLDisposeTokenUPP](#)) - input

Either a pointer to the token disposal function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myGetMarkTokenProc

myGetMarkTokenProc ([OSLGetMarkTokenUPP](#)) - input

Either a pointer to the function for returning a mark token provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myMarkProc

myMarkProc ([OSLMarkUPP](#)) - input

Either a pointer to the object-marking function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myAdjustMarksProc

myAdjustMarksProc ([OSLAdjustMarksUPP](#)) - input

Either a pointer to the mark-adjusting function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Parameter - myGetErrDescProc

myGetErrDescProc ([OSLGetErrDescUPP](#)) - input

Either a pointer to the error callback function provided by your application or NULL if no function is provided.

AESetObjectCallbacks Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

errAENotASpecialFunction

No error.

The handler pointer is odd.

There was not enough memory.

The keyword is invalid for a special function.

AESetObjectCallbacks - Parameters

myCompareProc ([OSLCompareUPP](#)) - input

Either a pointer to the object-comparison function provided by your application or NULL if no function is provided.

myCountProc ([OSLCountUPP](#)) - input

Either a pointer to the object-counting function provided by your application or NULL if no function is provided.

myDisposeTokenProc ([OSLDisposeTokenUPP](#)) - input

Either a pointer to the token disposal function provided by your application or NULL if no function is provided.

myGetMarkTokenProc ([OSLGetMarkTokenUPP](#)) - input

Either a pointer to the function for returning a mark token provided by your application or NULL if no function is provided.

myMarkProc ([OSLMarkUPP](#)) - input

Either a pointer to the object-marking function provided by your application or NULL if no function is provided.

myAdjustMarksProc ([OSLAdjustMarksUPP](#)) - input

Either a pointer to the mark-adjusting function provided by your application or NULL if no function is provided.

myGetErrDescProc ([OSLGetErrDescUPP](#)) - input

Either a pointer to the error callback function provided by your application or NULL if no function is provided.

rc ([OSErr](#)) - returns

Return code.

noErr

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

errAENotASpecialFunction

No error.

The handler pointer is odd.

There was not enough memory.

The keyword is invalid for a special function.

AESetObjectCallbacks - Remarks

This functions is not intended for use by OpenDoc part handlers.

Your application can provide only one each of the object callback functions specified by AESetObjectCallbacks: one object-comparison function, one object-counting function, and so on. As a result, each of these callback functions must perform the requested task (comparing, counting, and so on) for all the object classes that your application supports. In contrast, your application may provide many different object accessor functions if necessary, depending on the object classes and token types your application supports.

To replace object callback routines that have been previously installed, you can make another call to AESetObjectCallbacks. Each additional call to AESetObjectCallbacks replaces any object callback functions installed by previous calls to AESetObjectCallbacks. You cannot use this function to replace system object callback routines or object accessor functions. Only those routines you specify are replaced; to avoid replacing existing callback functions, specify a value of NULL for the functions you do not want to replace.

For information about writing object callback functions, see [Application-Defined Routines](#).

AESetObjectCallbacks - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

OSA Event Manager Functions

This chapter provides information about the OSA Event Manager functions. These functions are listed in alphabetic order.

The OSA Event Manager receives all semantic events. Each semantic-event aware application is registered with the OSA Event Manager through the `AEInit` method. The OSA Event Manager maintains a registry of running applications that have registered themselves. This registry contains the window handle, PID, and name of the application.

All intra-machine OSA events use an address descriptor containing the target's process ID (PID). The following code fragment demonstrates creating an address descriptor of this form:

```
AEAddressDesc theAddressDesc;  
PID           thePID;  
  
err = AECreatDesc (typePID, &thePID, sizeof(PID), &theAddressDesc);
```

An application can send an OSA event to itself using its own PID, which can be obtained by calling `DosGetInfoBlocks`; however, the preferred method for an application to send an event to itself is to generate an address descriptor using the predefined constant `kCurrentProcess`.

Events sent with a `typePID` address are delivered to the message queue registered by the application's [AEInit](#) call.

This release of the OS/2 OSA Event Manager does not support inter-machine communication. A future release will implement inter-machine events using SOM name service and security mechanisms.

The following lists OSA Event Manager functions in alphabetic order:

- [AEClearDesc](#)
- [AECoerceDesc](#)
- [AECoercePtr](#)
- [AECountItems](#)
- [AECreatDesc](#)
- [AECreatList](#)
- [AECreatOSAEvent](#)
- [AEDeleteItem](#)
- [AEDeleteKeyDesc](#)
- [AEDeleteParam](#)
- [AEDisposeDesc](#)
- [AEDuplicateDesc](#)
- [AEGetAppName](#)
- [AEGetArray](#)
- [AEGetAttributeDesc](#)
- [AEGetAttributePtr](#)
- [AEGetCoercionHandler](#)
- [AEGetDescData](#)
- [AEGetEventHandler](#)
- [AEGetHWND](#)
- [AEGetInteractionAllowed](#)
- [AEGetKeyDesc](#)
- [AEGetKeyPtr](#)
- [AEGetNthDesc](#)
- [AEGetNthPtr](#)
- [AEGetParamDesc](#)
- [AEGetParamPtr](#)
- [AEGetPID](#)
- [AEGetSpecialHandler](#)
- [AEGetTheCurrentEvent](#)
- [AEInit](#)

- [AEInstallCoercionHandler](#)
- [AEInstallEventHandler](#)
- [AEInstallSpecialHandler](#)
- [AELaunchApplication](#)
- [AEManagerInfo](#)
- [AEProcessOSAEEvent](#)
- [AEPutArray](#)
- [AEPutAttributeDesc](#)
- [AEPutAttributePtr](#)
- [AEPutDesc](#)
- [AEPutKeyDesc](#)
- [AEPutKeyPtr](#)
- [AEPutParamDesc](#)
- [AEPutParamPtr](#)
- [AEPutPtr](#)
- [AERemoveCoercionHandler](#)
- [AERemoveEventHandler](#)
- [AERemoveSpecialHandler](#)
- [AEResetTimer](#)
- [AEResumeTheCurrentEvent](#)
- [AESend](#)
- [AESetInteractionAllowed](#)
- [AESetTheCurrentEvent](#)
- [AESizeOfAttribute](#)
- [AESizeOfDescData](#)
- [AESizeOfKeyDesc](#)
- [AESizeOfNthItem](#)
- [AESizeOfParam](#)
- [AESuspendTheCurrentEvent](#)
- [AETerminate](#)
- [OSAInstallApplication](#) (C)
- [OSAInstallApplication](#) (OREXX)
- [OSAListApplications](#)
- [OSARemoveApplication](#) (C)
- [OSARemoveApplication](#) (OREXX)

AEClearDesc (OS/2)

AEClearDesc (OS/2) - Syntax

This function initializes the specified descriptor record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEDesc      *theAEDesc;
OSErr      rc;          /* Return code. */

rc = AEClearDesc(theAEDesc);
```

AEClearDesc (OS/2) Parameter - theAEDesc

theAEDesc ([AEDesc](#) *) - input

The address of the descriptor record to be initialized. The descriptor type in this structure is set to typeNull, and the data handle is set to NULL.

AEClearDesc (OS/2) Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

AEClearDesc (OS/2) - Parameters

theAEDesc ([AEDesc *](#)) - input
The address of the descriptor record to be initialized. The descriptor type in this structure is set to typeNull, and the data handle is set to NULL.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

AEClearDesc (OS/2) - Example Code

This example shows how to initialize a descriptor record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>
```

```
OSErr myErr;
AEDesc theDesc;
```

```
myErr = AEClearDesc(&theDesc);
```

AEClearDesc (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AECoerceDesc

AECoerceDesc - Syntax

This function coerces the data in a descriptor record to another descriptor type.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDesc      *theAEDesc;
DescType          toType;
AEDesc            *result;
OSErr             rc;          /* Return code. */

rc = AECoerceDesc(theAEDesc, toType, result);
```

AECoerceDesc Parameter - theAEDesc

theAEDesc (**const AEDesc ***) - input
The descriptor record whose data is to be coerced.

AECoerceDesc Parameter - toType

toType (**DescType**) - input
The desired descriptor type of the resulting descriptor record.

AECoerceDesc Parameter - result

result (**AEDesc ***) - output
The resulting descriptor record.

AECoerceDesc Return Value - rc

rc (**OSErr**) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate the descriptor record.

errAECocersionFail

Data could not be coerced to requested descriptor type.

AECocersionDesc - Parameters

theAECocersionDesc ([const AECocersionDesc *](#)) - input

The descriptor record whose data is to be coerced.

toType ([DescType](#)) - input

The desired descriptor type of the resulting descriptor record.

result ([AECocersionDesc *](#)) - output

The resulting descriptor record.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate the descriptor record.

errAECocersionFail

Data could not be coerced to requested descriptor type.

AECocersionDesc - Remarks

This function attempts to create a new descriptor record by coercing the specified descriptor record. Your application is responsible for using the [AEDisposeDesc](#) function to dispose of the resulting descriptor record once you are finished using it. If this function returns a nonzero result code, it returns a null descriptor record (a descriptor record of type typeNull, which does not contain any data) unless the OSA Event Manager is not available because of limited memory.

For a list of the descriptor types for which the OSA Event Manager provides coercions, see the table in section [Writing and Installing Coercion Handlers](#).

AECocersionDesc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AECocersionPtr

AECoercePtr - Syntax

This function coerces data to a desired descriptor type and, if successful, creates a descriptor record containing the newly coerced data.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

DescType    typeCode;
const void  *dataPtr;
Size        dataSize;
DescType    toType;
AEDesc      *result;
OSErr       rc;          /* Return code. */

rc = AECoercePtr(typeCode, dataPtr, dataSize,
                 toType, result);
```

AECoercePtr Parameter - typeCode

typeCode ([DescType](#)) - input
The descriptor type of the source data.

AECoercePtr Parameter - dataPtr

dataPtr (const void *) - input
A pointer to the data to be coerced.

AECoercePtr Parameter - dataSize

dataSize ([Size](#)) - input
The length, in bytes, of the data to be coerced.

AECoercePtr Parameter - toType

toType ([DescType](#)) - input
The desired descriptor type of the resulting descriptor record.

AECoercePtr Parameter - result

result ([AEDesc *](#)) - output
The resulting descriptor record.

AECoercePtr Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There was not enough memory to allocate the descriptor record.

errAECoercionFail
Data could not be coerced to the requested descriptor type.

AECoercePtr - Parameters

typeCode ([DescType](#)) - input
The descriptor type of the source data.

dataPtr (const void *) - input
A pointer to the data to be coerced.

dataSize ([Size](#)) - input
The length, in bytes, of the data to be coerced.

toType ([DescType](#)) - input
The desired descriptor type of the resulting descriptor record.

result ([AEDesc *](#)) - output
The resulting descriptor record.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There was not enough memory to allocate the descriptor record.

errAECoercionFail
Data could not be coerced to the requested descriptor type.

AECoercePtr - Remarks

This function creates a new descriptor record by coercing the specified data to a descriptor record of the specified descriptor type. You should use the [AEDisposeDesc](#) function to dispose of the resulting descriptor record once you are finished using it. If AECoercePtr returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

AECoeercePtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AECountItems

AECountItems - Syntax

This function counts the number of descriptor records in any descriptor list.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDescList    *theAEDescList;
long                *theCount;
OSErr               rc;          /* Return code. */

rc = AECountItems(theAEDescList, theCount);
```

AECountItems Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input
The descriptor list to be counted.

AECountItems Parameter - theCount

theCount ([long *](#)) - output
The number of descriptor records in the specified descriptor list.

AECountItems Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

errAENotAEDesc

The descriptor record was invalid.

AECountItems - Parameters

theAEDescList ([const AEDescList *](#)) - input
The descriptor list to be counted.

theCount (long *) - output
The number of descriptor records in the specified descriptor list.

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

errAENotAEDesc

The descriptor record was invalid.

AECountItems - Example Code

For an example of the use of AECountItems, see [Getting Data Out of a Descriptor List](#).

AECountItems - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

AECreateDesc

AECreateDesc - Syntax

This function converts data into a descriptor record.

```

#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

DescType    typeCode;
const void  *dataPtr;
Size        dataSize;
AEDesc      *result;
OSErr       rc;          /* Return code. */

rc = AECreatDesc(typeCode, dataPtr, dataSize,
                 result);

```

AECreatDesc Parameter - typeCode

typeCode ([DescType](#)) - input
The descriptor type for the descriptor record.

AECreatDesc Parameter - dataPtr

dataPtr (const void *) - input
A pointer to the data for the descriptor record.

AECreatDesc Parameter - dataSize

dataSize ([Size](#)) - input
The length, in bytes, of the data for the descriptor record.

AECreatDesc Parameter - result

result ([AEDesc *](#)) - output
The descriptor record that is created.

AECreatDesc Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate the descriptor record.

errAENotAEDesc

The result is not a valid descriptor record.

AECreatDesc - Parameters

typeCode ([DescType](#)) - input

The descriptor type for the descriptor record.

dataPtr (const void *) - input

A pointer to the data for the descriptor record.

dataSize ([Size](#)) - input

The length, in bytes, of the data for the descriptor record.

result ([AEDesc *](#)) - output

The descriptor record that is created.

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate the descriptor record.

errAENotAEDesc

The result is not a valid descriptor record.

AECreatDesc - Remarks

This function creates a new descriptor record that incorporates the specified data. Your application is responsible for using the [AEDisposeDesc](#) function to dispose of the resulting descriptor record when you no longer need it. You normally do this after receiving a result code from the [AESend](#) function. If this function returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

AECreatDesc - Example Code

For examples of the use of AECreatDesc, see [Adding Parameters to an OSA Event](#) and the figure in section [Creating an Address Descriptor Record](#).

AECreatDesc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Glossary](#)

AECreatelist

AECreatelist - Syntax

This function creates an empty descriptor list or AE record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const void    *factoringPtr;
Size          factoredSize; /* The size of the common data. */
BOOL          isRecord;
AEDescList    *resultList;
OSErr         rc;           /* Return code. */

rc = AECreatelist(factoringPtr, factoredSize,
                  isRecord, resultList);
```

AECreatelist Parameter - factoringPtr

factoringPtr (const void *) - input

A pointer to the data at the beginning of each descriptor that is the same for all descriptor records in the list. If there is no common data, or if you decide not to isolate the common data, specify NULL as the value of this parameter.

AECreatelist Parameter - factoredSize

factoredSize ([Size](#)) - input

The size of the common data.

If there is no common data, or if you decide not to isolate the common data, the value of this parameter must be 0.

AECreatelist Parameter - isRecord

isRecord ([BOOL](#)) - input

A flag indicating the kind of list to create.

TRUE

The OSA Event Manager creates an AE record.

FALSE

The OSA Event Manager creates a descriptor list.

AECreatelist Parameter - resultList

resultList ([AEDescList *](#)) - output
The descriptor list or AE record that is created.

AECreatelist Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

Parameter error (value of pointer is NULL or odd).

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate for the descriptor list.

errAENotAEDesc

The result is not a valid descriptor record.

AECreatelist - Parameters

factoringPtr (const void *) - input

A pointer to the data at the beginning of each descriptor that is the same for all descriptor records in the list. If there is no common data, or if you decide not to isolate the common data, specify NULL as the value of this parameter.

factoredSize ([Size](#)) - input

The size of the common data.

If there is no common data, or if you decide not to isolate the common data, the value of this parameter must be 0.

isRecord ([BOOL](#)) - input

A flag indicating the kind of list to create.

TRUE

The OSA Event Manager creates an AE record.

FALSE

The OSA Event Manager creates a descriptor list.

resultList ([AEDescList *](#)) - output

The descriptor list or AE record that is created.

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

Parameter error (value of pointer is NULL or odd).

ERROR_NOT_ENOUGH_MEMORY

There was not enough memory to allocate for the descriptor list.

errAENotAEDesc

The result is not a valid descriptor record.

AECreatelist - Remarks

This function creates an empty descriptor list or AE record. Your application is responsible for using the [AEDisposeDesc](#) function to dispose of the resulting descriptor record when you no longer need it. You normally do this after receiving a result code from the [AESend](#) function. If you intend to use a descriptor list for a factored OSA event array, you must provide, in the *factoredSize* parameter, a pointer to the data shared by all items in the array and, in the *factoredSize* parameter, the size of the common data. The common data must be 4, 8, or more than 8 bytes in length because it always consists of (a) the descriptor type (4 bytes); (b) the descriptor type (4 bytes) and the size of each item's data (4 bytes); or (c) the descriptor type (4 bytes), the size of each item's data (4 bytes), and some portion of the data itself (1 or more bytes). If this function returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

For information about data types used with OSA event arrays, see [OSA Event Array Data Types](#).

AECreatelist - Example Code

For an example of the use of AECreatelist, see the figure in section [Specifying Optional Parameters for an OSA Event](#).

AECreatelist - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AECreatOSAEvent

AECreatOSAEvent - Syntax

This function creates an OSA event with several important attributes but no parameters. Parameters are added to the OSA event after the event has been created.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEEEventClass theAEEEventClass;
AEEEventID theAEEEventID;
const AEAddressDesc *target;
short returnID;
long transactionID;
OSAEvent *result;
OSErr rc; /* Return code. */
```

```
rc = AECreatOSAEvent(theAEEEventClass, theAEEEventID,  
    target, returnID, transactionID, result);
```

AECreatOSAEvent Parameter - theAEEEventClass

theAEEEventClass ([AEEEventClass](#)) - input
The event class of the OSA event to be created.

AECreatOSAEvent Parameter - theAEEEventID

theAEEEventID ([AEEEventID](#)) - input
The event ID of the OSA event to be created.

AECreatOSAEvent Parameter - target

target ([const AEEAddressDesc *](#)) - input
The address of the server application.

AECreatOSAEvent Parameter - returnID

returnID (short) - input
The return ID for the OSA event. This parameter can also be set to the following constant:

kAutoGenerateReturnID
The OSA Event Manager assigns a return ID that is unique to the current process.

AECreatOSAEvent Parameter - transactionID

transactionID (long) - input
The transaction ID for the OSA event.

A *transaction* is a sequence of OSA events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All OSA events that are part of a transaction must have the same transaction ID. This parameter can also be set to the following constant:

kAnyTransactionID
No transaction ID is being used.

AECreatOSAEvent Parameter - result

result ([OSAEvent *](#)) - output
The OSA events that are created.

AECreatOSAEvent Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the OSA event.
errAENotAEDesc	The target descriptor record is invalid.

AECreatOSAEvent - Parameters

theAEEEventClass ([AEEEventClass](#)) - input
The event class of the OSA event to be created.

theAEEEventID ([AEEEventID](#)) - input
The event ID of the OSA event to be created.

target ([const AEAddressDesc *](#)) - input
The address of the server application.

returnID (short) - input
The return ID for the OSA event. This parameter can also be set to the following constant:

kAutoGenerateReturnID	The OSA Event Manager assigns a return ID that is unique to the current process.
-----------------------	--

transactionID (long) - input
The transaction ID for the OSA event.

A *transaction* is a sequence of OSA events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All OSA events that are part of a transaction must have the same transaction ID. This parameter can also be set to the following constant:

kAnyTransactionID	No transaction ID is being used.
-------------------	----------------------------------

result ([OSAEvent *](#)) - output
The OSA events that are created.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the OSA event.
errAENotAEDesc	

The target descriptor record is invalid.

AECreatOSAEvent - Remarks

This function creates an OSA event. Your application is responsible for using the [AEDisposeDesc](#) function to dispose of the OSA event when you no longer need it.

If this function returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

AECreatOSAEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEDeleteItem

AEDeleteItem - Syntax

This function deletes a descriptor record from a descriptor list. All subsequent descriptor records will then move up one place.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
OSErr               rc;                /* Return code. */

rc = AEDeleteItem(theAEDescList, index);
```

AEDeleteItem Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input

The descriptor list containing the descriptor record to be deleted.

AEDeleteItem Parameter - index

index (long) - input

The position of the descriptor record to delete (for example, 2 specifies the second item).

AEDeleteItem Return Value - rc

rc ([OSError](#)) - returns

Return code.

noErr

No error.

errAEDescNotFound

The descriptor record is not found.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEDeleteItem - Parameters

theAEDescList ([const AEDescList *](#)) - input

The descriptor list containing the descriptor record to be deleted.

index (long) - input

The position of the descriptor record to delete (for example, 2 specifies the second item).

rc ([OSError](#)) - returns

Return code.

noErr

No error.

errAEDescNotFound

The descriptor record is not found.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEDeleteItem - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

AEDeleteKeyDesc

AEDeleteKeyDesc - Syntax

This function deletes a keyword-specified descriptor record from an AE record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AERecord      *theAERecord;
AEKeyword      theAEKeyword;
OSErr          rc;          /* Return code. */

rc = AEDeleteKeyDesc(theAERecord, theAEKeyword);
```

AEDeleteKeyDesc Parameter - theAERecord

theAERecord ([AERecord *](#)) - input

The AE record containing the keyword-specified descriptor record to be deleted.

AEDeleteKeyDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the descriptor record to be deleted.

AEDeleteKeyDesc Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr	No error.
errAEDescNotFound	The descriptor record is not found.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.

AEDeleteKeyDesc - Parameters

theAERecord ([AERecord *](#)) - input

The AE record containing the keyword-specified descriptor record to be deleted.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the descriptor record to be deleted.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errAEDescNotFound

The descriptor record is not found.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEDeleteKeyDesc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

AEDeleteParam

AEDeleteParam - Syntax

This function deletes an OSA event parameter.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
OSErr               rc;          /* Return code. */

rc = AEDeleteParam(theOSAEvent, theAEKeyword);
```

AEDeleteParam Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event containing the parameter to be deleted.

AEDeleteParam Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the parameter to be deleted.

AEDeleteParam Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errAEDescNotFound

The descriptor record was not found.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEDeleteParam - Parameters

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event containing the parameter to be deleted.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the parameter to be deleted.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errAEDescNotFound

The descriptor record was not found.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEDeleteParam - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

AEDisposeDesc

AEDisposeDesc - Syntax

This function deallocates the memory used by a descriptor record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEDesc      *theAEDesc;
OSErr      rc;          /* Return code. */

rc = AEDisposeDesc(theAEDesc);
```

AEDisposeDesc Parameter - theAEDesc

theAEDesc ([AEDesc](#) *) - in/out

The descriptor record to deallocate. A null descriptor record is returned in this parameter. If you pass a null descriptor record in this parameter, noErr is returned.

AEDisposeDesc Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

AEDisposeDesc - Parameters

theAEDesc ([AEDesc](#) *) - in/out

The descriptor record to deallocate. A null descriptor record is returned in this parameter. If you pass a null descriptor record in this parameter, noErr is returned.

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

AEDisposeDesc - Remarks

For more information about using AEDisposeDesc, see [Disposing of OSA Event Data Structures](#).

AEDisposeDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEDuplicateDesc

AEDuplicateDesc - Syntax

This function makes a copy of a descriptor record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDesc      *theAEDesc;
AEDesc            *result;
OSErr             rc;          /* Return code. */

rc = AEDuplicateDesc(theAEDesc, result);
```

AEDuplicateDesc Parameter - theAEDesc

theAEDesc ([const AEDesc *](#)) - input
The descriptor record to be duplicated.

AEDuplicateDesc Parameter - result

result ([AEDesc *](#)) - output
The duplicate descriptor record that is created.

AEDuplicateDesc Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate the descriptor record.

AEDuplicateDesc - Parameters

theAEDesc ([const AEDesc *](#)) - input
The descriptor record to be duplicated.

result ([AEDesc *](#)) - output
The duplicate descriptor record that is created.

rc ([OSErr](#)) - returns
Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate the descriptor record.

AEDuplicateDesc - Remarks

This function creates a new descriptor record by copying the descriptor record from the *theAEDesc* parameter. Your application is responsible for using the [AEDisposeDesc](#) function to dispose of the resulting descriptor record when you no longer need it. You normally do this after receiving a result code from the [AESend](#) function. If this function returns a nonzero result code, a null descriptor record is returned. It is common for applications to send OSA events that have one or more attributes or parameters in common. For example, if you send a series of OSA events to the same application, the address attribute is the same. In these cases, the most efficient way to create the necessary OSA events is to make a template OSA event that you can then copy-by calling this function-as needed. You then fill in or change the remaining parameters and attributes of the copy, send the copy by calling [AESend](#), and dispose of the copy-by calling [AEDisposeDesc](#)-after [AESend](#) returns a result code.

AEDuplicateDesc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AEGetAppName (OS/2)

AEGetAppName (OS/2) - Syntax

This function returns the name of an application that sent an OSA event. It is used by scripting components during the processing of recorded text events.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

PID      thePID;
char     *appName;
ULONG    *size;
OSErr    rc;          /* Return code. */

rc = AEGetAppName(thePID, appName, size);
```

AEGetAppName (OS/2) Parameter - thePID

thePID ([PID](#)) - input
The process ID whose application name is to be returned.

AEGetAppName (OS/2) Parameter - appName

appName (char *) - input
A pointer to a buffer in which the name of the application is to be returned. The name that is returned is the same as that registered by the [AEInit](#) method called from this process. This parameter can be passed as NULL to determine the size of the buffer required to hold the application name, in which case the *size* parameter returns the required buffer length.

AEGetAppName (OS/2) Parameter - size

size ([ULONG](#) *) - in/out
On input, if *appName* is not NULL, this parameter specifies the size of the buffer pointed to by the *appName* parameter. On output, this parameter is updated to reflect the actual number of characters placed in the buffer.

AEGetAppName (OS/2) Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
bufferIsSmall	The size of the buffer is not large enough to hold the data to be returned.
procNotFound	There is no eligible process with the specified process serial number.
errAAppNotFound	The specified application or part handler is not found in the registration data base.

AEGetAppName (OS/2) - Parameters

thePID ([PID](#)) - input
The process ID whose application name is to be returned.

appName (char *) - input
A pointer to a buffer in which the name of the application is to be returned. The name that is returned is the same as that registered by the [AEInit](#) method called from this process. This parameter can be passed as NULL to determine the size of the buffer required to hold the application name, in which case the *size* parameter returns the required buffer length.

size ([ULONG *](#)) - in/out
On input, if *appName* is not NULL, this parameter specifies the size of the buffer pointed to by the *appName* parameter. On output, this parameter is updated to reflect the actual number of characters placed in the buffer.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
bufferIsSmall	The size of the buffer is not large enough to hold the data to be returned.
procNotFound	There is no eligible process with the specified process serial number.
errAAppNotFound	The specified application or part handler is not found in the registration data base.

AEGetAppName (OS/2) - Remarks

The application must be registered with the OSA Event Manager before calling this function. Applications are registered using the [AEInit](#) function.

If the buffer passed in the *appName* parameter is too small, an error is returned. If the buffer is larger than the required size, only the number of bytes returned in the *size* parameter is written to it.

AEGetAppName (OS/2) - Example Code

In this example, the event handler extracts the PID of the sender application from the event received and calls the `AEGetAppName` method with this PID to determine the name of the application.

```
#define INCL_OSAEVENTS
```



```

#define INCL_OSAAPI
#include <os2.h>

OSErr APIENTRY MyEventHandler(const OSAEvent *theOSAEvent,
                             const OSAEvent *reply,
                             long handlerRefcon);

{
    OSERR    myErr;                /* result code */
    PID      senderAddr;          /* PID of sender of event */
    DescType actualType;          /* actual address type */
    Size      addrSize;           /* size of sender's address */
    ULONG     nameSize;           /* size of sender application's name */
    PSZ       appName;            /* name of sender's application */

    myErr = AEGetAttributePtr(theOSAEvent,
                             keyAddressAttr,
                             typePID,
                             &actualType,
                             &senderAddr,
                             sizeof(senderAddr),
                             &addrSize);

    if(myErr == noErr)
    {
        myErr = AEGetAppName(senderAddr, NULL, &nameSize);

        if(myErr == noErr)
        {
            appName = (PSZ) malloc(nameSize);
            myErr = AEGetAppName(senderAddr, appName, &nameSize);

            /* .... Handle the event ... */
        }
    }

    return(myErr);
}

```

AEGetAppName (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetArray

AEGetArray - Syntax

This function converts an OSA event array to the corresponding C array and places the converted array in a specified buffer.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDescList    *theAEDescList;
AEArrayType          arrayType;
AEArrayDataPointer  arrayPtr;
Size                maximumSize;
DescType            *itemType;
Size                *itemSize;
long                *itemCount;
OSErr               rc;          /* Return code. */

rc = AEGetArray(theAEDescList, arrayType,
                arrayPtr, maximumSize, itemType, itemSize,
                itemCount);
```

AEGetArray Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input

A descriptor list containing the desired array. If the array is of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor list must be factored.

AEGetArray Parameter - arrayType

arrayType ([AEArrayType](#)) - input

The OSA event array type to be converted. This is specified by one of the following constants:

- kAEDataArray
- kAEPackedArray
- kAEHandleArray
- kAEDescArray
- kAEKeyDescArray

AEGetArray Parameter - arrayPtr

arrayPtr ([AEArrayDataPointer](#)) - input

A pointer to the buffer for storing the array.

AEGetArray Parameter - maximumSize

maximumSize ([Size](#)) - input
The maximum length, in bytes, of the buffer for storing the array.

AEGetArray Parameter - itemType

itemType ([DescType](#) *) - output
The descriptor type of the returned array items (for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray).

AEGetArray Parameter - itemSize

itemSize ([Size](#) *) - output
The size (in bytes) of the returned array items (for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray).

AEGetArray Parameter - itemCount

itemCount (long *) - output
The number of items in the resulting array.

AEGetArray Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the array of descriptor types.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetArray - Parameters

theAEDescList ([const AEDescList *](#)) - input

A descriptor list containing the desired array. If the array is of type [kAEDataArray](#), [kAEPackedArray](#), or [kAEHandleArray](#), the descriptor list must be factored.

arrayType ([AEArrayType](#)) - input

The OSA event array type to be converted. This is specified by one of the following constants:

[kAEDataArray](#)

[kAEPackedArray](#)

[kAEHandleArray](#)

[kAEDescArray](#)

[kAEKeyDescArray](#)

arrayPtr ([AEArrayDataPointer](#)) - input

A pointer to the buffer for storing the array.

maximumSize ([Size](#)) - input

The maximum length, in bytes, of the buffer for storing the array.

itemType ([DescType *](#)) - output

The descriptor type of the returned array items (for arrays of type [kAEDataArray](#), [kAEPackedArray](#), or [kAEHandleArray](#)).

itemSize ([Size *](#)) - output

The size (in bytes) of the returned array items (for arrays of type [kAEDataArray](#), [kAEPackedArray](#), or [kAEHandleArray](#)).

itemCount (long *) - output

The number of items in the resulting array.

rc ([OSErr](#)) - returns

Return code.

[noErr](#)

No error.

[ERROR_NOT_ENOUGH_MEMORY](#)

There is not enough memory to allocate for the array of descriptor types.

[errAEWrongDataType](#)

A wrong descriptor type is specified.

[errAENotAEDesc](#)

The descriptor record is invalid.

[errAEReplyNotArrived](#)

The reply has not yet arrived.

AEGetArray - Remarks

An OSA event array is an array created by calling the [AEPutArray](#) function and stored in a descriptor list.

The AEGetArray function uses a buffer identified by the pointer in the *arrayPtr* parameter to return the converted data for the OSA event array specified by the *theAEDescList* parameter. Even if the descriptor list that contains the array is factored, the converted data for each array item includes the data common to all the descriptor records in the list. The OSA Event Manager automatically reconstructs the common data for each item when you call AEGetArray.

For more information about data types and constants used with AEGetArray, see [OSA Event Array Data Types](#).

AEGetArray - Topics

Select an item:

[Syntax](#)

AEGetAttributeDesc

AEGetAttributeDesc - Syntax

This function returns the descriptor record for a specified OSA event attribute.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
DescType            desiredType;
AEDesc              *result;
OSErr               rc;          /* Return code. */

rc = AEGetAttributeDesc(theOSAEvent, theAEKeyword,
                        desiredType, result);
```

AEGetAttributeDesc Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired attribute.

AEGetAttributeDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired attribute. This parameter can be set to any of the following values:

- keyAddressAttr Address of target or client application
- keyEventClassAttr Event class
- keyEventIDAttr Event ID
- keyEventSourceAttr Nature of source application
- keyInteractLevelAttr

Settings to allow the OSA Event Manager to bring server application to the foreground

keyMissedKeywordAttr
First required parameter remaining in OSA event

keyOptionalKeywordAttr
List of optional parameters for OSA event

keyOriginalAddressAttr
Address of original source of OSA event.

keyReturnIDAttr
Return ID for reply OSA event

keyTimeoutAttr
Length of time in ticks that client will wait for reply or result from the server

keyTransactionIDAttr
Transaction ID identifying a series of OSA event

AEGetAttributeDesc Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type for the descriptor record to be returned; if the requested OSA event attribute is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is typeWildcard, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event attribute.

AEGetAttributeDesc Parameter - result

result ([AEDesc *](#)) - output

A copy of the descriptor record from the desired attribute coerced to the descriptor type specified by the *desiredType* parameter.

AEGetAttributeDesc Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate the descriptor record.

errAECOercionFail
Data could not be coerced to the requested descriptor type.

errAEDescNotFound
The descriptor record is not found.

errAENotAEDesc
The descriptor record is invalid.

errAEReplyNotArrived
The reply has not yet arrived.

AEGetAttributeDesc - Parameters

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event containing the desired attribute.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the desired attribute. This parameter can be set to any of the following values:

keyAddressAttr

Address of target or client application

keyEventClassAttr

Event class

keyEventIDAttr

Event ID

keyEventSourceAttr

Nature of source application

keyInteractLevelAttr

Settings to allow the OSA Event Manager to bring server application to the foreground

keyMissedKeywordAttr

First required parameter remaining in OSA event

keyOptionalKeywordAttr

List of optional parameters for OSA event

keyOriginalAddressAttr

Address of original source of OSA event.

keyReturnIDAttr

Return ID for reply OSA event

keyTimeoutAttr

Length of time in ticks that client will wait for reply or result from the server

keyTransactionIDAttr

Transaction ID identifying a series of OSA event

desiredType ([DescType](#)) - input

The desired descriptor type for the descriptor record to be returned; if the requested OSA event attribute is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event attribute.

result ([AEDesc *](#)) - output

A copy of the descriptor record from the desired attribute coerced to the descriptor type specified by the *desiredType* parameter.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate the descriptor record.

errAECOercionFail

Data could not be coerced to the requested descriptor type.

errAEDescNotFound

The descriptor record is not found.

errAENotAEDesc

The descriptor record is invalid.

errAERReplyNotArrived

The reply has not yet arrived.

AEGetAttributeDesc - Remarks

This function returns, in the result parameter, the descriptor record for the OSA event attribute with the specified keyword. Your application should call the [AEDisposeDesc](#) function to dispose of the resulting descriptor record after your application has finished using it. If this function

returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

AEGetAttributeDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEGetAttributePtr

AEGetAttributePtr - Syntax

This function returns a pointer to a buffer that contains the data from a specified OSA event attribute.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
DescType            desiredType;
DescType            *typeCode;
void                *dataPtr;
Size                maximumSize;
Size                *actualSize;
OSErr               rc;          /* Return code. */

rc = AEGetAttributePtr(theOSAEvent, theAEKeyword,
                      desiredType, typeCode, dataPtr, maximumSize,
                      actualSize);
```

AEGetAttributePtr Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired attribute.

AEGetAttributePtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the desired attribute. This parameter can be set to any of the following values:

keyAddressAttr	Address of target or client application
keyEventClassAttr	Event class
keyEventIDAttr	Event ID
keyEventSourceAttr	Nature of source application
keyInteractLevelAttr	Settings to allow the OSA Event Manager to bring server application to the foreground
keyMissedKeywordAttr	First required parameter remaining in OSA event
keyOptionalKeywordAttr	List of optional parameters for OSA event
keyOriginalAddressAttr	Address of original source of OSA event.
keyReturnIDAttr	Return ID for reply OSA event
keyTimeoutAttr	Length of time in ticks that client will wait for reply or result from the server
keyTransactionIDAttr	Transaction ID identifying a series of OSA event

AEGetAttributePtr Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type for the data to be returned; if the requested OSA event attribute is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is typeWildCard, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event attribute.

AEGetAttributePtr Parameter - typeCode

typeCode ([DescType](#) *) - output

The descriptor type of the returned data.

AEGetAttributePtr Parameter - dataPtr

dataPtr - output

A pointer to the buffer in which the returned data is stored.

AEGetAttributePtr Parameter - maximumSize

maximumSize ([Size](#)) - input

The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

AEGetAttributePtr Parameter - actualSize

actualSize ([Size *](#)) - input

The length, in bytes, of the data for the specified OSA event attribute. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the attribute was returned.

AEGetAttributePtr Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the buffer.

errAECoercionFail

Data could not be coerced to the requested descriptor type.

errAEDescNotFound

The descriptor record could not found.

errAENotAEDesc

The descriptor record is not valid.

errAEReplyNotArrived

The reply has not yet arrived.

AEGetAttributePtr - Parameters

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event containing the desired attribute.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the desired attribute. This parameter can be set to any of the following values:

keyAddressAttr

Address of target or client application

keyEventClassAttr

Event class

keyEventIDAttr

Event ID

keyEventSourceAttr

Nature of source application

keyInteractLevelAttr
Settings to allow the OSA Event Manager to bring server application to the foreground

keyMissedKeywordAttr
First required parameter remaining in OSA event

keyOptionalKeywordAttr
List of optional parameters for OSA event

keyOriginalAddressAttr
Address of original source of OSA event.

keyReturnIDAttr
Return ID for reply OSA event

keyTimeoutAttr
Length of time in ticks that client will wait for reply or result from the server

keyTransactionIDAttr
Transaction ID identifying a series of OSA event

desiredType ([DescType](#)) - input

The desired descriptor type for the data to be returned; if the requested OSA event attribute is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildCard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event attribute.

typeCode ([DescType](#) *) - output

The descriptor type of the returned data.

dataPtr - output

A pointer to the buffer in which the returned data is stored.

maximumSize ([Size](#)) - input

The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

actualSize ([Size](#) *) - input

The length, in bytes, of the data for the specified OSA event attribute. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the attribute was returned.

rc ([OSError](#)) - returns

Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the buffer.

errAECoercionFail
Data could not be coerced to the requested descriptor type.

errAEDescNotFound
The descriptor record could not found.

errAENotAEDesc
The descriptor record is not valid.

errAEReplyNotArrived
The reply has not yet arrived.

AEGetAttributePtr - Remarks

This function uses a buffer to return the data from an OSA event attribute with the specified keyword, which it attempts to coerce to the descriptor type specified by the *desiredType* parameter.

AEGetAttributePtr - Example Code

For an example of the use of the AEGetAttributePtr function, see [Getting Data Out of an Attribute](#) and [Writing OSA Event Handlers](#).

AEGetAttributePtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetCoercionHandler

AEGetCoercionHandler - Syntax

This function returns the handler for a specified descriptor type coercion.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

DescType      fromType;
DescType      toType;
AECoercionHandlerUPP *handler;
long          *handlerRefcon;
BOOL          *fromTypeIsDesc;
BOOL          isSysHandler;
OSErr         rc;          /* Return code. */

rc = AEGetCoercionHandler(fromType, toType,
                          handler, handlerRefcon, fromTypeIsDesc,
                          isSysHandler);
```

AEGetCoercionHandler Parameter - fromType

fromType ([DescType](#)) - input
The descriptor type of the data coerced by the handler.

AEGetCoercionHandler Parameter - toType

toType ([DescType](#)) - input
The descriptor type of the resulting data.

AEGetCoercionHandler Parameter - handler

handler ([AECoercionHandlerUPP *](#)) - output
A pointer to the desired coercion handler.

AEGetCoercionHandler Parameter - handlerRefcon

handlerRefcon (long *) - output
The reference constant for the desired handler. The OSA Event Manager passes this reference constant to the handler each time the handler is called.

AEGetCoercionHandler Parameter - fromTypesDesc

fromTypesDesc ([BOOL *](#)) - output
Pointer to a flag indicating whether the coercion handler expects the data to be passed as a descriptor record.

TRUE	The coercion handler expects the data to be passed as a descriptor record.
FALSE	The coercion handler expects a pointer to the data.

AEGetCoercionHandler Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input
Specifies the coercion table from which to get the handler.

TRUE	The handler is taken from the system coercion table.
FALSE	The handler is taken from the application coercion table.

AEGetCoercionHandler Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the buffer.
errAEHandlerNotFound	

No coercion handler is found.
errAESysHandlerNotLoaded
The system handler could not be loaded.

AEGetCoercionHandler - Parameters

fromType ([DescType](#)) - input

The descriptor type of the data coerced by the handler.

toType ([DescType](#)) - input

The descriptor type of the resulting data.

handler ([AECoercionHandlerUPP *](#)) - output

A pointer to the desired coercion handler.

handlerRefcon (long *) - output

The reference constant for the desired handler. The OSA Event Manager passes this reference constant to the handler each time the handler is called.

fromTypesDesc ([BOOL *](#)) - output

Pointer to a flag indicating whether the coercion handler expects the data to be passed as a descriptor record.

TRUE

The coercion handler expects the data to be passed as a descriptor record.

FALSE

The coercion handler expects a pointer to the data.

isSysHandler ([BOOL](#)) - input

Specifies the coercion table from which to get the handler.

TRUE

The handler is taken from the system coercion table.

FALSE

The handler is taken from the application coercion table.

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the buffer.

errAEHandlerNotFound

No coercion handler is found.

errAESysHandlerNotLoaded

The system handler could not be loaded.

AEGetCoercionHandler - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

AEGetDescData (OS/2)

AEGetDescData (OS/2) - Syntax

This function returns the descriptor type of the specified descriptor record and a copy of the data that it references.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDesc      *theAEDesc;
DescType          *typeCode;
Ptr               dataBuffer;
Size              maximumSize;
Size              *actualSize;
OSErr             rc;          /* Return code. */

rc = AEGetDescData(theAEDesc, typeCode, dataBuffer,
                  maximumSize, actualSize);
```

AEGetDescData (OS/2) Parameter - theAEDesc

theAEDesc (*const AEDesc **) - input
The address of the descriptor record whose data is to be returned.

AEGetDescData (OS/2) Parameter - typeCode

typeCode (*DescType **) - output
A pointer to the descriptor type of the specified descriptor record. This parameter can be passed as NULL to indicate that data for this parameter is not to be returned.

AEGetDescData (OS/2) Parameter - dataBuffer

dataBuffer (*Ptr*) - input
A pointer to a buffer into which the descriptor data is to be copied.

AEGetDescData (OS/2) Parameter - maximumSize

maximumSize (*Size*) - input
The size, in bytes, of the buffer pointed to by the *dataBuffer* parameter.

AEGetDescData (OS/2) Parameter - actualSize

actualSize ([Size *](#)) - output

A pointer to the actual size, in bytes, of the data copied into the buffer pointed to by the *dataBuffer* parameter. This parameter can be passed as NULL to indicate that data for this parameter is not to be returned.

AEGetDescData (OS/2) Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

AEGetDescData (OS/2) - Parameters

theAEDesc ([const AEDesc *](#)) - input

The address of the descriptor record whose data is to be returned.

typeCode ([DescType *](#)) - output

A pointer to the descriptor type of the specified descriptor record. This parameter can be passed as NULL to indicate that data for this parameter is not to be returned.

dataBuffer ([Ptr](#)) - input

A pointer to a buffer into which the descriptor data is to be copied.

maximumSize ([Size](#)) - input

The size, in bytes, of the buffer pointed to by the *dataBuffer* parameter.

actualSize ([Size *](#)) - output

A pointer to the actual size, in bytes, of the data copied into the buffer pointed to by the *dataBuffer* parameter. This parameter can be passed as NULL to indicate that data for this parameter is not to be returned.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

AEGetDescData (OS/2) - Remarks

The data is copied into the specified buffer up to the maximum number of bytes given in the *maximumSize* parameter. The actual number of bytes copied is returned in the *actualSize* parameter.

AEGetDescData (OS/2) - Example Code

This example uses the `AEGetDescData` method to extract the PID out of an address descriptor record. It assumes *theOSAEvent* has been created and initialized correctly.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

OSErr    myErr;                                /* result code */
OSAEvent theOSAEvent;                          /* an OSA event */
AEAddressDesc theAddrDesc;                    /* address descriptor record */
DescType actualType;                          /* actual descriptor type of the data */
PID      thePID;                              /* process ID */

myErr = AEGetAttributeDesc(theOSAEvent,
                           keyAddressAttr,
                           typePID,
                           &theAddrDesc);

if(myErr == noErr)
{
    myErr = AEGetDescData(&theAddrDesc,
                         &actualType,
                         &thePID,
                         sizeof(thePID),
                         &actualSize);
}
```

AEGetDescData (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetEventHandler

AEGetEventHandler - Syntax

This function gets an entry from an OSA event dispatch table.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEEventClass    theAEEventClass;
```

```

AEEventID          theAEEventID;
AEEventHandlerUPP *handler;
long               *handlerRefcon;
BOOL               isSysHandler;
OSErr              rc;                /* Return code. */

rc = AEGGetEventHandler(theAEEventClass, theAEEventID,
                       handler, handlerRefcon, isSysHandler);

```

AEGetEventHandler Parameter - theAEEventClass

theAEEventClass ([AEEventClass](#)) - input

The value of the event class field of the dispatch table entry for the desired handler.

AEGetEventHandler Parameter - theAEEventID

theAEEventID ([AEEventID](#)) - input

The value of the event ID field of the dispatch table entry for the desired handler.

AEGetEventHandler Parameter - handler

handler ([AEEventHandlerUPP *](#)) - output

The AEGetEventHandler function returns, in this parameter, a pointer to the specified handler.

AEGetEventHandler Parameter - handlerRefcon

handlerRefcon (long *) - output

The AEGetEventHandler function returns, in this parameter, the reference constant from the dispatch table entry for the specified handler.

AEGetEventHandler Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating from which OSA event dispatch table the handler is to be returned.

TRUE

The handler from the system dispatch table is returned.

FALSE

The handler from your application's dispatch table is returned.

AEGetEventHandler Return Value - rc

rc (**OSErr**) - returns
Return code.

noErr
No error.

errAEHandlerNotFound
A handler is not found for an OSA event.

errAESysHandlerNotLoaded
The system handler could not be loaded.

AEGetEventHandler - Parameters

theAEEEventClass (**AEEEventClass**) - input
The value of the event class field of the dispatch table entry for the desired handler.

theAEEEventID (**AEEEventID**) - input
The value of the event ID field of the dispatch table entry for the desired handler.

handler (**AEEEventHandlerUPP ***) - output
The AEGetEventHandler function returns, in this parameter, a pointer to the specified handler.

handlerRefcon (long *) - output
The AEGetEventHandler function returns, in this parameter, the reference constant from the dispatch table entry for the specified handler.

isSysHandler (**BOOL**) - input
A flag indicating from which OSA event dispatch table the handler is to be returned.

TRUE
The handler from the system dispatch table is returned.

FALSE
The handler from your application's dispatch table is returned.

rc (**OSErr**) - returns
Return code.

noErr
No error.

errAEHandlerNotFound
A handler is not found for an OSA event.

errAESysHandlerNotLoaded
The system handler could not be loaded.

AEGetEventHandler - Remarks

This function returns, in the handler parameter, a pointer to the handler for the OSA event dispatch table entry you specify in the *theAEEEventClass* and *theAEEEventID* parameters. You can use the *typeWildcard* constant for either or both of these parameters; however, AEGetEventHandler returns an error unless an entry exists that specifies *typeWildcard* in exactly the same way. For example, if you specify *typeWildcard* in both the *theAEEEventClass* and *theAEEEventID* parameters, the OSA Event Manager will not return the first handler for any event class and event ID in the dispatch table; instead, the dispatch table must contain an entry that specifies *typeWildcard* for both the event class and the event ID.

AEGetEventHandler - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEGetHWND (OS/2)

AEGetHWND (OS/2) - Syntax

This function returns the window handle from a specified PID of an application that has been registered through the [AEInit](#) function.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

PID          thePID;
ULONG        *windowHandle;
OSErr        rc;          /* Return code. */

rc = AEGetHWND(thePID, windowHandle);
```

AEGetHWND (OS/2) Parameter - thePID

thePID ([PID](#)) - input

The process ID registered with the [AEInit](#) function.

AEGetHWND (OS/2) Parameter - windowHandle

windowHandle ([ULONG *](#)) - output

A pointer to the window handle.

AEGetHWND (OS/2) Return Value - rc

rc ([OSError](#)) - returns
Return code.

noErr

No error.

procNotFound

There are no eligible process with the specified process serial number.

AEGetHWND (OS/2) - Parameters

thePID ([PID](#)) - input

The process ID registered with the [AEInit](#) function.

windowHandle ([ULONG *](#)) - output

A pointer to the window handle.

rc ([OSError](#)) - returns
Return code.

noErr

No error.

procNotFound

There are no eligible process with the specified process serial number.

AEGetHWND (OS/2) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

AEGetInteractionAllowed

AEGetInteractionAllowed - Syntax

This function returns the current user interaction preferences for responding to an OSA event.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>
```

```
AEInteractAllowed    *level;
OSError              rc;    /* Return code. */
```

```
rc = AEGetInteractionAllowed(level);
```

AEGetInteractionAllowed Parameter - level

level ([AEInteractAllowed *](#)) - output

The current user interaction level. This parameter can be set to one of the following values, indicating the user interaction preferences for responding to an OSA event:

kAEInteractWithSelf

This flag indicates that the server application may interact with the user in response to an OSA event only when the client application and server application are the same-that is, only when the application is sending the OSA event to itself.

kAEInteractWithLocal

This flag indicates that the server application may interact with the user in response to an OSA event only if the client application is on the same computer as the server application. This is the default if your application has not used the [AESetInteractionAllowed](#) to set the interaction level explicitly.

kAEInteractWithALL

This flag indicates that the server application may interact with the user in response to an OSA event sent from any client application on any computer.

AEGetInteractionAllowed Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

AEGetInteractionAllowed - Parameters

level ([AEInteractAllowed *](#)) - output

The current user interaction level. This parameter can be set to one of the following values, indicating the user interaction preferences for responding to an OSA event:

kAEInteractWithSelf

This flag indicates that the server application may interact with the user in response to an OSA event only when the client application and server application are the same-that is, only when the application is sending the OSA event to itself.

kAEInteractWithLocal

This flag indicates that the server application may interact with the user in response to an OSA event only if the client application is on the same computer as the server application. This is the default if your application has not used the [AESetInteractionAllowed](#) to set the interaction level explicitly.

kAEInteractWithALL

This flag indicates that the server application may interact with the user in response to an OSA event sent from any client application on any computer.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

AEGetInteractionAllowed - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AEGetKeyDesc

AEGetKeyDesc - Syntax

This function returns the descriptor record for a keyword-specified descriptor record. This function can be used to get a descriptor record out of an AE record or an OSA event record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AERecord      *theAERecord;
AEKeyword      theAEKeyword;
DescType      desiredType;
AEDesc        *result;
OSErr          rc;          /* Return code. */

rc = AEGetKeyDesc(theAERecord, theAEKeyword,
                  desiredType, result);
```

AEGetKeyDesc Parameter - theAERecord

theAERecord ([AERecord](#) *) - input
The AE record containing the desired descriptor record.

AEGetKeyDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired descriptor record.

AEGetKeyDesc Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type for the descriptor record to be returned. If the requested descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned descriptor record is the same as the descriptor type of the original descriptor record.

AEGetKeyDesc Parameter - result

result ([AEDesc *](#)) - output

A copy of the keyword-specified descriptor record, coerced to the descriptor type specified in the *desiredType* parameter.

AEGetKeyDesc Return Value - rc

rc ([OSErr](#)) - returns

Return code.

`noErr`

No error.

`ERROR_NOT_ENOUGH_MEMORY`

There is not enough memory to allocate to the descriptor record.

`errAECoercionFail`

Data could not be coerced to the requested descriptor type.

`errAEDescNotFound`

The descriptor record is not found.

`errAENotAEDesc`

The descriptor record is invalid.

`errAEReplyNotArrived`

The reply has not yet arrived.

AEGetKeyDesc - Parameters

theAERecord ([AERecord *](#)) - input

The AE record containing the desired descriptor record.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the desired descriptor record.

desiredType ([DescType](#)) - input

The desired descriptor type for the descriptor record to be returned. If the requested descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned descriptor record is the same as the descriptor type of the original descriptor record.

result ([AEDesc *](#)) - output

A copy of the keyword-specified descriptor record, coerced to the descriptor type specified in the *desiredType* parameter.

rc ([OSErr](#)) - returns

Return code.

`noErr`

No error.

`ERROR_NOT_ENOUGH_MEMORY`

There is not enough memory to allocate to the descriptor record.

errAECoercionFail	Data could not be coerced to the requested descriptor type.
errAEDescNotFound	The descriptor record is not found.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetKeyDesc - Remarks

This function returns a copy of the descriptor record for a keyword-specified descriptor record. Your application should call the [AEDisposeDesc](#) function to dispose of the resulting descriptor record after your application has finished using it. If AEGetKeyDesc returns a nonzero result code, it returns a descriptor record of descriptor type typeNull. A descriptor record of this type does not contain any data.

AEGetKeyDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEGetKeyPtr

AEGetKeyPtr - Syntax

This function returns a pointer to a buffer that contains the data from a keyword-specified descriptor record. This function can be used to get data from an AE record or an OSA event record.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AERecord      *theAERecord;
AEKeyword      theAEKeyword;
DescType      desiredType;
DescType      *typeCode;
void          *dataPtr;
Size          maximumSize;
Size          *actualSize;
OSErr         rc;          /* Return code. */

rc = AEGetKeyPtr(theAERecord, theAEKeyword,
                desiredType, typeCode, dataPtr, maximumSize,
                actualSize);
```

AEGetKeyPtr Parameter - theAERecord

theAERecord ([AERecord *](#)) - input
The AE record containing the desired data.

AEGetKeyPtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired descriptor record.

AEGetKeyPtr Parameter - desiredType

desiredType ([DescType](#)) - input
The desired descriptor type for the data to be returned; if the requested data is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is typeWildcard, no coercion is performed, and the descriptor type of returned data is the same as the descriptor type of the original data.

AEGetKeyPtr Parameter - typeCode

typeCode ([DescType *](#)) - output
The descriptor type of the returned data.

AEGetKeyPtr Parameter - dataPtr

dataPtr - output
A pointer to the buffer for storing the data.

AEGetKeyPtr Parameter - maximumSize

maximumSize ([Size](#)) - input
The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

AEGetKeyPtr Parameter - actualSize

actualSize ([Size *](#)) - output

The length, in bytes, of the data for the keyword-specified descriptor record. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the parameter was returned.

AEGetKeyPtr Return Value - rc

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory to allocate for the buffer.

errAECoectionFail

Data could not be coerced to the requested descriptor type.

errAEDescNotFound

The descriptor record is not found.

errAEWrongDataType

A wrong descriptor type is specified.

errAENotAEDesc

The descriptor record is invalid.

errAEReplyNotArrived

The reply has not yet arrived.

AEGetKeyPtr - Parameters

theAERecord ([AERecord *](#)) - input

The AE record containing the desired data.

theAEKeyword ([AEKeyword](#)) - input

The keyword that specifies the desired descriptor record.

desiredType ([DescType](#)) - input

The desired descriptor type for the data to be returned; if the requested data is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of returned data is the same as the descriptor type of the original data.

typeCode ([DescType *](#)) - output

The descriptor type of the returned data.

dataPtr - output

A pointer to the buffer for storing the data.

maximumSize ([Size](#)) - input

The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

actualSize ([Size *](#)) - output

The length, in bytes, of the data for the keyword-specified descriptor record. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the parameter was returned.

rc ([OSError](#)) - returns

Return code.

noErr

	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the buffer.
errAECoercionFail	Data could not be coerced to the requested descriptor type.
errAEDescNotFound	The descriptor record is not found.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetKeyPtr - Remarks

This function uses a buffer to return the data from a keyword-specified OSA event parameter, which the function attempts to coerce to the descriptor type specified by the *desiredType* parameter.

AEGetKeyPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEGetNthDesc

AEGetNthDesc - Syntax

This function returns a copy of a descriptor record from any descriptor list or [AERecord](#) structure.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
DescType            desiredType;
AEKeyword            *theAEKeyword;
AEDesc              *result;
OSErr               rc;                /* Return code. */

rc = AEGetNthDesc(theAEDescList, index, desiredType,
                  theAEKeyword, result);
```

AEGetNthDesc Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input
The descriptor list containing the desired descriptor record.

AEGetNthDesc Parameter - index

index (long) - input
The position of the desired descriptor record in the list (for example, 2 specifies the second descriptor record).

AEGetNthDesc Parameter - desiredType

desiredType ([DescType](#)) - input
The desired descriptor type for the copy of the descriptor record to be returned. If the desired descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the copied descriptor record is the same as the descriptor type of the original descriptor record.

AEGetNthDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword *](#)) - output
The keyword of the specified descriptor record, if you are getting data from a list of keyword-specified descriptor records; otherwise, the value `typeWildcard` is returned.

AEGetNthDesc Parameter - result

result ([AEDesc *](#)) - output
A copy of the desired descriptor record coerced to the descriptor type specified by the *desiredType* parameter.

AEGetNthDesc Return Value - rc

rc ([OSErr](#)) - returns
Return code.

`noErr` No error.

ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the descriptor record.
errAECoercionFail	Data could not be coerced to the requested descriptor type.
errAEDescNotFound	The descriptor record is not found.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetNthDesc - Parameters

theAEDescList ([const AEDescList *](#)) - input

The descriptor list containing the desired descriptor record.

index (long) - input

The position of the desired descriptor record in the list (for example, 2 specifies the second descriptor record).

desiredType ([DescType](#)) - input

The desired descriptor type for the copy of the descriptor record to be returned. If the desired descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the copied descriptor record is the same as the descriptor type of the original descriptor record.

theAEKeyword ([AEKeyword *](#)) - output

The keyword of the specified descriptor record, if you are getting data from a list of keyword-specified descriptor records; otherwise, the value `typeWildcard` is returned.

result ([AEDesc *](#)) - output

A copy of the desired descriptor record coerced to the descriptor type specified by the *desiredType* parameter.

rc ([OSError](#)) - returns

Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate for the descriptor record.
errAECoercionFail	Data could not be coerced to the requested descriptor type.
errAEDescNotFound	The descriptor record is not found.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetNthDesc - Remarks

This function returns a specified descriptor record from a specified descriptor list. Your application should call the [AEDisposeDesc](#) function to dispose of the resulting descriptor record after your application has finished using it. If `AEGetNthDesc` returns a nonzero result code, it returns a descriptor record of descriptor type `typeNull`. A descriptor record of this type does not contain any data.

AEGetNthDesc - Topics

Select an item:

[Syntax](#)

AEGetNthPtr

AEGetNthPtr - Syntax

This function returns a pointer to a buffer that contains a copy of a descriptor record from any descriptor list.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
DescType            desiredType;
AEKeyword            *theAEKeyword;
DescType            *typeCode;
void                *dataPtr;
Size                maximumSize;
Size                *actualSize;
OSErr               rc;                /* Return code. */

rc = AEGetNthPtr(theAEDescList, index, desiredType,
                 theAEKeyword, typeCode, dataPtr, maximumSize,
                 actualSize);
```

AEGetNthPtr Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input
The descriptor list containing the desired descriptor record.

AEGetNthPtr Parameter - index

index (long) - input
The position of the desired descriptor record in the list (for example, 2 specifies the second descriptor record).

AEGetNthPtr Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type for the copy of the descriptor record to be returned. If the desired descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the copied descriptor record is the same as the descriptor type of the original descriptor record.

AEGetNthPtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#) *) - output

The keyword of the specified descriptor record, if you are getting data from a list of keyword-specified descriptor records; otherwise, the value `typeWildcard` is returned.

AEGetNthPtr Parameter - typeCode

typeCode ([DescType](#) *) - output

The descriptor type of the returned descriptor record.

AEGetNthPtr Parameter - dataPtr

dataPtr - output

A pointer to the buffer in which the returned descriptor record is stored.

AEGetNthPtr Parameter - maximumSize

maximumSize ([Size](#)) - input

The maximum length, in bytes, of the data to be returned. You allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

AEGetNthPtr Parameter - actualSize

actualSize ([Size](#) *) - output

The length, in bytes, of the data for the specified descriptor record. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the descriptor record was returned.

AEGetNthPtr Return Value - rc

rc (**OSErr**) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the buffer.

errAECoercionFail
Data could not be coerced to the requested descriptor type.

errAEDescNotFound
The descriptor record is not found.

errAEWrongDataType
The wrong descriptor type is specified.

errAENotAEDesc
The descriptor record is specified.

errAEReplyNotArrived
The reply has not yet arrived.

AEGetNthPtr - Parameters

theAEDescList (**const AEDescList ***) - input
The descriptor list containing the desired descriptor record.

index (**long**) - input
The position of the desired descriptor record in the list (for example, 2 specifies the second descriptor record).

desiredType (**DescType**) - input
The desired descriptor type for the copy of the descriptor record to be returned. If the desired descriptor record is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the copied descriptor record is the same as the descriptor type of the original descriptor record.

theAEKeyword (**AEKeyword ***) - output
The keyword of the specified descriptor record, if you are getting data from a list of keyword-specified descriptor records; otherwise, the value `typeWildcard` is returned.

typeCode (**DescType ***) - output
The descriptor type of the returned descriptor record.

dataPtr - output
A pointer to the buffer in which the returned descriptor record is stored.

maximumSize (**Size**) - input
The maximum length, in bytes, of the data to be returned. You allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

actualSize (**Size ***) - output
The length, in bytes, of the data for the specified descriptor record. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the descriptor record was returned.

rc (**OSErr**) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate for the buffer.

errAECoercionFail
Data could not be coerced to the requested descriptor type.

errAEDescNotFound
The descriptor record is not found.

errAEWrongDataType
The wrong descriptor type is specified.

errAENotAEDesc
The descriptor record is specified.

errAEReplyNotArrived
The reply has not yet arrived.

AEGetNthPtr - Remarks

This function uses a buffer to return a specified descriptor record from a specified descriptor list; the function attempts to coerce the descriptor record to the descriptor type specified by the *desiredType* parameter.

AEGetNthPtr - Example Code

For an example of the use of AEGetNthPtr, see the figure in section [Getting Data Out of a Descriptor List](#).

AEGetNthPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetParamDesc

AEGetParamDesc - Syntax

This function returns the descriptor record for a specified OSA event parameter.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
DescType            desiredType;
AEDesc              *result;
OSErr               rc;          /* Return code. */

rc = AEGetParamDesc(theOSAEvent, theAEKeyword,
                    desiredType, result);
```

AEGetParamDesc Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired parameter.

AEGetParamDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

AEGetParamDesc Parameter - desiredType

desiredType ([DescType](#)) - input
The desired descriptor type for the descriptor record to be returned. If the requested OSA event parameter is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event parameter.

AEGetParamDesc Parameter - result

result ([AEDesc *](#)) - output
The descriptor record from the desired OSA event parameter coerced to the descriptor type specified in *desiredType*.

AEGetParamDesc Return Value - rc

rc ([OSErr](#)) - returns
Return code.

<code>noErr</code>	No error.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	There is not enough memory to allocate for the descriptor record.
<code>errAECoercionFail</code>	Data could not be coerced to the requested descriptor type.
<code>errAEDescNotFound</code>	Descriptor record is not found.
<code>errAENotAEDesc</code>	The descriptor record is invalid.
<code>errAEReplyNotArrived</code>	The reply has not yet arrived.

AEGetParamDesc - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired parameter.

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

desiredType ([DescType](#)) - input
The desired descriptor type for the descriptor record to be returned. If the requested OSA event parameter is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event parameter.

result ([AEDesc *](#)) - output
The descriptor record from the desired OSA event parameter coerced to the descriptor type specified in *desiredType*.

rc ([OSErr](#)) - returns
Return code.

<code>noErr</code>	No error.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	There is not enough memory to allocate for the descriptor record.
<code>errAECoercionFail</code>	Data could not be coerced to the requested descriptor type.
<code>errAEDescNotFound</code>	Descriptor record is not found.
<code>errAENotAEDesc</code>	The descriptor record is invalid.
<code>errAEReplyNotArrived</code>	The reply has not yet arrived.

AEGetParamDesc - Remarks

This function returns, in the result parameter, the descriptor record for a specified OSA event parameter, which it attempts to coerce to the descriptor type specified by the *desiredType* parameter. An application should call the [AEDisposeDesc](#) function to dispose of the resulting descriptor record after your application has finished using it.

If `AEGetParamDesc` returns a nonzero result code, it returns a null descriptor record unless the OSA Event Manager is not available because of limited memory.

AEGetParamDesc - Example Code

For an example of the use of `AEGetParamDesc`, see [Getting Data Out of an OSA Event Parameter](#).

AEGetParamDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetParamPtr

AEGetParamPtr - Syntax

This function returns a pointer to a buffer that contains the data from a specified OSA event parameter.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
DescType            desiredType;
DescType            *typeCode;
void                *dataPtr;
Size                maximumSize;
Size                *actualSize;
OSErr               rc;          /* Return code. */

rc = AEGetParamPtr(theOSAEvent, theAEKeyword,
                  desiredType, typeCode, dataPtr, maximumSize,
                  actualSize);
```

AEGetParamPtr Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired parameter.

AEGetParamPtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

AEGetParamPtr Parameter - desiredType

desiredType ([DescType](#)) - input
The desired descriptor type for the data to be returned; if the requested OSA event parameter is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event parameter.

AEGetParamPtr Parameter - typeCode

typeCode (**DescType** *) - output
The descriptor type of the returned data.

AEGetParamPtr Parameter - dataPtr

dataPtr - output
A pointer to the buffer in which the returned data is stored.

AEGetParamPtr Parameter - maximumSize

maximumSize (**Size**) - input
The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

AEGetParamPtr Parameter - actualSize

actualSize (**Size** *) - output
The length, in bytes, of the data for the specified OSA event parameter. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the parameter was returned.

AEGetParamPtr Return Value - rc

rc (**OSErr**) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the buffer.
errAECoeercionFail	Data could not be coerced to the requested descriptor type.
errAEDescNotFound	The descriptor record is not found.
errAEWrongDataType	The wrong descriptor type was specified.
errAENotAEDesc	The descriptor record is invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AEGetParamPtr - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event containing the desired parameter.

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

desiredType ([DescType](#)) - input
The desired descriptor type for the data to be returned; if the requested OSA event parameter is not of this type, the OSA Event Manager attempts to coerce it to this type. If the value of this parameter is `typeWildcard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the OSA event parameter.

typeCode ([DescType *](#)) - output
The descriptor type of the returned data.

dataPtr - output
A pointer to the buffer in which the returned data is stored.

maximumSize ([Size](#)) - input
The maximum length, in bytes, of the data to be returned. You must allocate at least this amount of storage for the buffer specified by the *dataPtr* parameter.

actualSize ([Size *](#)) - output
The length, in bytes, of the data for the specified OSA event parameter. If this value is larger than the value of the *maximumSize* parameter, not all of the data for the parameter was returned.

rc ([OSErr](#)) - returns
Return code.

<code>noErr</code>	No error.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	There is not enough memory to allocate the buffer.
<code>errAECoercionFail</code>	Data could not be coerced to the requested descriptor type.
<code>errAEDescNotFound</code>	The descriptor record is not found.
<code>errAEWrongDataType</code>	The wrong descriptor type was specified.
<code>errAENotAEDesc</code>	The descriptor record is invalid.
<code>errAEReplyNotArrived</code>	The reply has not yet arrived.

AEGetParamPtr - Remarks

This function uses a buffer to return the data from a specified OSA event parameter, which it attempts to coerce to the descriptor type specified by the *desiredType* parameter.

AEGetParamPtr - Example Code

For examples of the use of `AEGetParamPtr`, see [Getting Data Out of an OSA Event](#).

AEGetParamPtr - Topics

Select an item:
[Syntax](#)

AEGetPID (OS/2)

AEGetPID (OS/2) - Syntax

This function returns the process ID (PID) of the specified application.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

char      *appName;
PID       *thePID;
OSErr     rc;      /* Return code. */

rc = AEGetPID(appName, thePID);
```

AEGetPID (OS/2) Parameter - appName

appName (char *) - input

A pointer to a buffer containing the application name. This value must match the name used by the application in its [AEInit](#) call.

AEGetPID (OS/2) Parameter - thePID

thePID (PID *) - output

A pointer to the process ID.

AEGetPID (OS/2) Return Value - rc

rc (OSErr) - returns
Return code.

noErr

No error.

ERROR_PROC_NOT_FOUND

AEGetPID (OS/2) - Parameters

appName (char *) - input

A pointer to a buffer containing the application name. This value must match the name used by the application in its [AEInit](#) call.

thePID (PID *) - output

A pointer to the process ID.

rc (OSError) - returns

Return code.

noErr

No error.

ERROR_PROC_NOT_FOUND

The process is unknown to the OSA Event Manager.

AEGetPID (OS/2) - Remarks

This method determines if an application is currently running and is registered as an OSA event-aware application with the OSA Event Manager—that is, the application has called the [AEInit](#) method.

All inter-machine OSA events use an address descriptor containing the target's process ID (PID). The following code fragment demonstrates creating an address descriptor of this form:

```
AEAddressDesc theAddressDesc;  
PID            thePID;
```

```
err = AECreatDesc (typePID, &thePID, sizeof(PID), &theAddressDesc);
```

An application can send an OSA event to itself using its own PID, which can be obtained by calling `DosGetInfoBlocks`; however, the preferred method for an application to send an event to itself is to generate an address descriptor using the predefined constant `kCurrentProcess`.

Events sent with a typePID address are delivered to the message queue registered by the application's [AEInit](#) call.

AEGetPID (OS/2) - Example Code

This example returns the PID of an application.

```
#define INCL_OSAEVENTS
```

```
#define INCL_OSAAPI
```

```
#include <os2.h>
```

```
OSError myErr; /* result code */
```

```
PID thePID; /* process ID of application */
```

```
char appName[] = "My App"; /* name of application, as registered with  
OSAInstallApplication */
```

```
myErr = AEGetPID(appName, &thePID);
```

AEGetPID (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEGetSpecialHandler

AEGetSpecialHandler - Syntax

This function returns the specified special handler.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEKeyword      functionClass;
UniversalProcPtr *handler;
BOOL           isSysHandler;
OSErr          rc;          /* Return code. */

rc = AEGetSpecialHandler(functionClass, handler,
                        isSysHandler);
```

AEGetSpecialHandler Parameter - functionClass

functionClass ([AEKeyword](#)) - input

The keyword for the special handler that is installed. This parameter can be set to one of the following values:

- | | |
|-----------------------|---|
| keyPreDispatch | Handler with the same parameters as an OSA event handler that is called immediately before the OSA Event Manager dispatches an OSA event. |
| keyAECountProc | Object-counting function. |
| keyAECompateProc | Object-comparison function. |
| keyAEDisposeTokenProc | Token disposal function. |
| keyAEGetErrDescProc | Error callback function. |

keyAEMarkTokenProc
Mark token function.

keyAEMarkProc
Object-marking function.

keyAEAdjustMarksProc
Mark-adjusting function.

AEGetSpecialHandler Parameter - handler

handler ([UniversalProcPtr](#) *) - output
A pointer to the special handler.

AEGetSpecialHandler Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input
A flag indicating from which special handler dispatch table the handler is to be taken.

TRUE
The handler is taken from the system special handler dispatch table.

FALSE
The handler is taken from the application's special handler dispatch table.

AEGetSpecialHandler Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the handler.
errAENotASpecialFunction	A wrong keyword for a special handler was specified, or no handler of that type is installed.
errAESysHandlerNotLoaded	The system handler could not be loaded.

AEGetSpecialHandler - Parameters

functionClass ([AEKeyword](#)) - input
The keyword for the special handler that is installed. This parameter can be set to one of the following values:

keyPreDispatch
Handler with the same parameters as an OSA event handler that is called immediately before the OSA Event

Manager dispatches an OSA event.

keyAECountProc
Object-counting function.

keyAECompateProc
Object-comparison function.

keyAEDisposeTokenProc
Token disposal function.

keyAEGetErrDescProc
Error callback function.

keyAEMarkTokenProc
Mark token function.

keyAEMarkProc
Object-marking function.

keyAEAdjustMarksProc
Mark-adjusting function.

handler ([UniversalProcPtr](#) *) - output
A pointer to the special handler.

isSysHandler ([BOOL](#)) - input
A flag indicating from which special handler dispatch table the handler is to be taken.

TRUE
The handler is taken from the system special handler dispatch table.

FALSE
The handler is taken from the application's special handler dispatch table.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

ERROR_NOT_ENOUGH_MEMORY
There is not enough memory to allocate the handler.

errAENotASpecialFunction
A wrong keyword for a special handler was specified, or no handler of that type is installed.

errAESysHandlerNotLoaded
The system handler could not be loaded.

AEGetSpecialHandler - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AEGetTheCurrentEvent

AEGetTheCurrentEvent - Syntax

This function returns the OSA event that is currently being handled.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

OSAEvent      *theOSAEvent;
OSErr         rc;          /* Return code. */

rc = AEGGetCurrentEvent(theOSAEvent);
```

AEGetTheCurrentEvent Parameter - theOSAEvent

theOSAEvent ([OSAEvent](#) *) - input

The OSA event that is currently being handled. If no OSA event is currently being handled, a null descriptor record is returned.

AEGetTheCurrentEvent Return Value - rc

rc ([OSErr](#)) - returns

Return code.

[noErr](#)

No error.

AEGetTheCurrentEvent - Parameters

theOSAEvent ([OSAEvent](#) *) - input

The OSA event that is currently being handled. If no OSA event is currently being handled, a null descriptor record is returned.

rc ([OSErr](#)) - returns

Return code.

[noErr](#)

No error.

AEGetTheCurrentEvent - Remarks

In many applications, the handling of an OSA event involves one or more long chains of calls to internal routines. This function makes it unnecessary for these calls to include the current OSA event as a parameter; the routines can simply call `AEGetTheCurrentEvent` to get the current OSA event when it is needed. You can also use the `AEGetTheCurrentEvent` function to make sure that no OSA event is currently being handled. For example, suppose your application always uses an application-defined routine to delete a file. That routine can first call `AEGetTheCurrentEvent` and delete the file only if `AEGetTheCurrentEvent` returns a null descriptor record (that is, only if no OSA event is currently being handled).

AEGetTheCurrentEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEInit (OS/2)

AEInit (OS/2) - Syntax

This function registers a semantic-event aware application with the OSA Event Manager.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

ULONG      windowHandle;
char       *appName;
BOOL       acceptRemoteEvents;
OSErr      rc;          /* Return code. */

rc = AEInit(windowHandle, appName, acceptRemoteEvents);
```

AEInit (OS/2) Parameter - windowHandle

windowHandle ([ULONG](#)) - input
A PM window handle of the window procedure to receive all OSA events.

AEInit (OS/2) Parameter - appName

appName (char *) - input
The descriptive name of the application. This name should be the same as the one used in the [OSAInstallApplication](#) method, if applicable.

AEInit (OS/2) Parameter - acceptRemoteEvents

acceptRemoteEvents (BOOL) - input

This parameter is currently ignored by the OS/2 OSA Event Manager. A future release of OSA will support inter-machine OSA events. This parameter will then be used to control whether the application accepts remote events.

AEInit (OS/2) Return Value - rc

rc (OSError) - returns

Return code.

noErr

No error.

errAESystemError

A general scripting system error occurred.

AEInit (OS/2) - Parameters

windowHandle (ULONG) - input

A PM window handle of the window procedure to receive all OSA events.

appName (char *) - input

The descriptive name of the application. This name should be the same as the one used in the [OSAInstallApplication](#) method, if applicable.

acceptRemoteEvents (BOOL) - input

This parameter is currently ignored by the OS/2 OSA Event Manager. A future release of OSA will support inter-machine OSA events. This parameter will then be used to control whether the application accepts remote events.

rc (OSError) - returns

Return code.

noErr

No error.

errAESystemError

A general scripting system error occurred.

AEInit (OS/2) - Remarks

This function is used by all OSA-aware applications to register a window handle that will receive all OSA events.

Because all OSA components are implemented as ring 3 DLLs, they enter on the thread of the calling application. The OSA Event Manager will issue DosGetInfoBlocks to obtain the PID of the caller. All internal data structures, such as those that contain the entry points of OSA event handlers, are managed by the PID.

This method should be called immediately after the WinCreateStandardWindow function to identify the window procedure which is to receive semantic events and before any other PM calls are invoked.

For more information about the Win* functions and PM messages, see *Presentation Manager Programming Reference* .

AEInit (OS/2) - Example Code

This example shows how to register an application with the OSA Event Manager.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#define INCL_WIN
#include <os2.h>

#define APPNAME "My App"          /* Application Name, as registered with
OSAInstallApplication */

/* Window procedure prototype */
MRESULT EXPENTRY WindowProc(HWND hwnd, MSGID msg, MPARAM mp1, MPARAM mp2);

int main(int argc, char *argv[])
{
    HAB hab;                      /* anchor block handle */
    HMQ hmq;                      /* message queue handle */
    HWND hwnd;                   /* client window handle */
    HWND hwndFrame;              /* frame window handle */
    QMSG qmsg;                   /* message from queue */
    char szClassName[] = "myclassname"; /* window class name */
    ULONG flClassStyle = 0;       /* window class style */
    USHORT usExtra = 0;           /* extra storage per window */
    OSERR myErr;                  /* result code */

    hab = WinInitialize();
    hmq = WinCreateMsgQueue(hab, 0);
    WinRegisterClass(hab,
                    szClassName,
                    WindowProc,
                    flClassStyle,
                    usExtra);
    hwndFrame = WinCreateStdWindow(HWND_DESKTOP, ..., &hwnd);

    myErr = AEInit(hwnd, APPNAME, FALSE);
    if(myErr) return myErr;

    while(WinGetMsg(hab, &qmsg, (HWND)0, 0, 0))
    {
        WinDispatch(hab, &qmsg);
    }

    ...
}
```

AEInit (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

AEInstallCoercionHandler (OS/2)

AEInstallCoercionHandler (OS/2) - Syntax

This function installs a coercion handler routine in the application's coercion handler dispatch table.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

DescType      fromType;
DescType      toType;
AECoercionHandlerUPP handler;
long          handlerRefcon;
BOOL          fromTypeIsDesc;
BOOL          isSysHandler;
OSErr         rc;          /* Return code. */

rc = AEInstallCoercionHandler(fromType, toType,
                             handler, handlerRefcon, fromTypeIsDesc,
                             isSysHandler);
```

AEInstallCoercionHandler (OS/2) Parameter - fromType

fromType ([DescType](#)) - input
The descriptor type of the data coerced by the handler.

AEInstallCoercionHandler (OS/2) Parameter - toType

toType ([DescType](#)) - input
The descriptor type of the resulting data. If there was already an entry in the specified coercion handler table for the same source descriptor type and result descriptor type, the existing entry is replaced.

AEInstallCoercionHandler (OS/2) Parameter - handler

handler ([AECoercionHandlerUPP](#)) - input
A pointer to the coercion handler. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler

entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

AEInstallCoercionHandler (OS/2) Parameter - handlerRefcon

handlerRefcon (long) - input

A reference constant passed by the OSA Event Manager to the handler each time the handler is called. If your handler does not expect a reference constant, this parameter should be set to 0.

AEInstallCoercionHandler (OS/2) Parameter - fromTypeIsDesc

fromTypeIsDesc (BOOL) - input

The form of the data to be coerced. This parameter can be set to one of the following values:

TRUE

The coercion handler expects the data to be passed as a descriptor record.

FALSE

The coercion handler expects a pointer to the data. Because this is more efficient for the OSA Event Manager to provide a pointer to data rather than to a descriptor record, all coercion routines should accept a pointer to the data if possible.

AEInstallCoercionHandler (OS/2) Parameter - isSysHandler

isSysHandler (BOOL) - input

A flag indicating whether to add the handler to the system coercion table.

TRUE

The OSA Event Manager adds the handler to the system coercion table making it available to all applications.

FALSE

The OSA Event Manager adds the handler to the application's coercion table.

AEInstallCoercionHandler (OS/2) Return Value - rc

rc (OSError) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The value of the handler is NULL or odd.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAESystemError

A general scripting system error occurred.

AEInstallCoercionHandler (OS/2) - Parameters

fromType ([DescType](#)) - input

The descriptor type of the data coerced by the handler.

toType ([DescType](#)) - input

The descriptor type of the resulting data. If there was already an entry in the specified coercion handler table for the same source descriptor type and result descriptor type, the existing entry is replaced.

handler ([AECoectionHandlerUPP](#)) - input

A pointer to the coercion handler. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

handlerRefcon (long) - input

A reference constant passed by the OSA Event Manager to the handler each time the handler is called. If your handler does not expect a reference constant, this parameter should be set to 0.

fromTypesDesc ([BOOL](#)) - input

The form of the data to be coerced. This parameter can be set to one of the following values:

TRUE

The coercion handler expects the data to be passed as a descriptor record.

FALSE

The coercion handler expects a pointer to the data. Because this is more efficient for the OSA Event Manager to provide a pointer to data rather than to a descriptor record, all coercion routines should accept a pointer to the data if possible.

isSysHandler ([BOOL](#)) - input

A flag indicating whether to add the handler to the system coercion table.

TRUE

The OSA Event Manager adds the handler to the system coercion table making it available to all applications.

FALSE

The OSA Event Manager adds the handler to the application's coercion table.

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The value of the handler is NULL or odd.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAESystemError

A general scripting system error occurred.

AEInstallCoercionHandler (OS/2) - Remarks

Before using this method to install a handler for a particular descriptor type into the system coercion handler dispatch table, use the [AEGetCoercionHandler](#) method to determine whether the table already contains a coercion handler for that description type. If an entry exists, the [AEGetCoercionHandler](#) function returns a reference constant and a pointer to that handler. Chain the coercion handler by providing, in the *handlerRefcon* parameter of this method, pointers to the previous handler and its reference constant. If the coercion handler returns the error `errAECoectionFail`, pointers should be used to call the previous handler. If the system coercion handler was removed, be sure to reinstall the chained handlers.

AEInstallCoercionHandler (OS/2) - Example Code

This example shows how to install coercion handlers in the application and system tables. Both handlers coerce a date in long format into the application-defined date string.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

/* Application-defined descriptor type */
#define typeMyDateString    0x65746164    /* "date" */

/* Prototype for application coercion handler */
OSErr APIENTRY MyApplicationCoercionHandler(DescType typeCode,
                                           const void *dataPtr,
                                           Size dataSize,
                                           DescType toType,
                                           long handlerRefcon,
                                           AEDesc *result);

OSErr myErr;
AESystemHandler sysCoercionHandler;          /* system handler structure
char procedureName[] = "MySystemCoercionHandler"; /* name of coercion handle
char moduleName[]    = "mymodule.dll";    /* module where the handler resides

sysCoercionHandler.procedureName = procedureName;
sysCoercionHandler.moduleName = moduleName;

/* Install in application table */
myErr = AEInstallCoercionHandler(typeLongDateTime,    /* input type */
                                typeMyDateString,    /* result type */
                                (AEC coercionHandlerUPP)MyApplicationCoercion
                                0,                    /* reference constant */
                                FALSE,                /* input is a pointer to data */
                                FALSE);               /* install in application table

/* Install in system table */
myErr = AEInstallCoercionHandler(typeLongDateTime, /* input type */
                                typeMyDateString, /* result type */
                                (AEC coercionHandlerUPP)&sysCoercionHandler,
                                0,                    /* reference constant */
                                FALSE,                /* input is a pointer to data */
                                TRUE);               /* install in system table */
```

AEInstallCoercionHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEInstallEventHandler (OS/2)

AEInstallEventHandler (OS/2) - Syntax

This function adds an entry to the application's OSA event dispatch table.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEEEventClass      theAEEEventClass;
AEEEventID         theAEEEventID;
AEEEventHandlerUPP handler;
long               handlerRefcon;
BOOL               isSysHandler;
OSErr              rc;          /* Return code. */

rc = AEInstallEventHandler(theAEEEventClass,
                          theAEEEventID, handler, handlerRefcon,
                          isSysHandler);
```

AEInstallEventHandler (OS/2) Parameter - theAEEEventClass

theAEEEventClass ([AEEEventClass](#)) - input
The event class for the OSA event or events to be dispatched.

AEInstallEventHandler (OS/2) Parameter - theAEEEventID

theAEEEventID ([AEEEventID](#)) - input
The event ID for the OSA event or events to be dispatched.

AEInstallEventHandler (OS/2) Parameter - handler

handler ([AEEEventHandlerUPP](#)) - input
A pointer to an OSA event handler for this dispatch table entry. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

AEInstallEventHandler (OS/2) Parameter - handlerRefcon

handlerRefcon (long) - input

The reference constant that is passed by the OSA Event Manager to the handler each time the handler is called. If your handler does not use a reference constant, this parameter should be set to 0.

AEInstallEventHandler (OS/2) Parameter - isSysHandler

isSysHandler (BOOL) - input

A flag indicating whether to add the handler to the system OSA event dispatch table.

TRUE

The OSA Event Manager passes the handler to the system OSA event dispatch table making it available to all applications.

FALSE

The OSA Event Manager passes the handler to the application's OSA event dispatch table. The OSA Event Manager searches the application's dispatch table first; the system dispatch table is searched only if the necessary handler is not found in your application's dispatch table.

AEInstallEventHandler (OS/2) Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The system handler flag is NULL or odd.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAESystemError

A general scripting system error occurred.

AEInstallEventHandler (OS/2) - Parameters

theAEEEventClass (AEEEventClass) - input

The event class for the OSA event or events to be dispatched.

theAEEEventID (AEEEventID) - input

The event ID for the OSA event or events to be dispatched.

handler (AEEEventHandlerUPP) - input

A pointer to an OSA event handler for this dispatch table entry. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

handlerRefcon (long) - input

The reference constant that is passed by the OSA Event Manager to the handler each time the handler is called. If your handler does not use a reference constant, this parameter should be set to 0.

isSysHandler (BOOL) - input

A flag indicating whether to add the handler to the system OSA event dispatch table.


```

OSErr myErr;
AESystemHandler sysEventHandler; /* system handler structure
char procedureName[] = "MyOpenDocSysHandler"; /* name of event handler */
char moduleName[] = "mymodule.dll"; /* module where the handler resides */

sysEventHandler.procedureName = procedureName;
sysEventHandler.moduleName = moduleName;

/* Install in application table */
myErr = AEInstallEventHandler( kCoreEventClass, /* event class */
                              kAEOpenDocuments, /* event ID */
                              (AEEEventHandlerUPP)MyOpenDocAppHandler,
                              0, /* reference constant */
                              FALSE); /* install in application table

/* Install in system table */
myErr = AEInstallEventHandler( kCoreEventClass, /* event class */
                              kAEOpenDocuments, /* event ID */
                              (AEEEventHandlerUPP)&sysEventHandler,
                              0, /* reference constant */
                              TRUE); /* install in system table

```

AEInstallEventHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEInstallSpecialHandler (OS/2)

AEInstallSpecialHandler (OS/2) - Syntax

This function installs a special handler in the application's special handler dispatch table.

```

#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

AEKeyword      functionClass;
UniversalProcPtr handler;
BOOL           isSysHandler;
OSErr          rc; /* Return code. */

rc = AEInstallSpecialHandler(functionClass,

```



```
handler, isSysHandler);
```

AEInstallSpecialHandler (OS/2) Parameter - functionClass

functionClass ([AEKeyword](#)) - input

A keyword for the special handler that is installed. This parameter can be set to any of the following constants:

keyPreDispatch

Handler with the same parameters as the OSA event handler called immediately before the OSA Event Manager dispatches an OSA event.

Note: If there was already an entry in the specified special handler dispatch table for the same value, it is replaced with this parameter.

keyAECCountProc

Object-counting function.

keyAECompareProc

Object-comparison function.

keyDisposeTokenProc

Token disposal function.

keyAEGetErrDescProc

Error callback function.

keyAEMarkTokenProc

Mark token function.

keyAEAdjustMarkProc

Mark-adjusting function.

AEInstallSpecialHandler (OS/2) Parameter - handler

handler ([UniversalProcPtr](#)) - input

A pointer to the special handler. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handler is being installed; therefore, a cast must be used to pass this parameter.

AEInstallSpecialHandler (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to add the handler to the system special handler dispatch table.

TRUE

The OSA Event Manager adds the handler to the system special handler dispatch table making it available to all applications.

FALSE

The OSA Event Manager adds the handler to the application's special handler dispatch table.

AEInstallSpecialHandler (OS/2) Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
paramErr	The handler pointer is NULL or odd.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAENotASpecialFunction	A wrong keyword for special function is specified.
errAESystemError	A general scripting system error occurred.

AEInstallSpecialHandler (OS/2) - Parameters

functionClass ([AEKeyword](#)) - input

A keyword for the special handler that is installed. This parameter can be set to any of the following constants:

keyPreDispatch	Handler with the same parameters as the OSA event handler called immediately before the OSA Event Manager dispatches an OSA event. Note: If there was already an entry in the specified special handler dispatch table for the same value, it is replaced with this parameter.
keyAECountProc	Object-counting function.
keyAECompareProc	Object-comparison function.
keyDisposeTokenProc	Token disposal function.
keyAEGetErrDescProc	Error callback function.
keyAEMarkTokenProc	Mark token function.
keyAEAdjustMarkProc	Mark-adjusting function.

handler ([UniversalProcPtr](#)) - input

A pointer to the special handler. The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handler is being installed; therefore, a cast must be used to pass this parameter.

isSysHandler ([BOOL](#)) - input

A flag indicating whether to add the handler to the system special handler dispatch table.

TRUE	The OSA Event Manager adds the handler to the system special handler dispatch table making it available to all applications.
FALSE	The OSA Event Manager adds the handler to the application's special handler dispatch table.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
paramErr	The handler pointer is NULL or odd.
ERROR_NOT_ENOUGH_MEMORY	

errAENotASpecialFunction There is not enough memory.
errAESystemError A wrong keyword for special function is specified.
 A general scripting system error occurred.

AEInstallSpecialHandler (OS/2) - Example Code

This example shows how to install, in both the application and system tables, a system special handler to do predispatch of events.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

/* Prototype for application predispatch handler */
OSErr APIENTRY MyPreDispatchHandler(const OSAEvent *theOSAEvent,
                                     OSAEvent *reply,
                                     long handlerRefcon);

OSErr myErr;
AESystemHandler sysSpecialHandler; /* system handler structure
char procedureName[] = "MyPreDispatchSysHandler"; /* name of special handler
char moduleName[] = "mymodule.dll"; /* module where the handler resides

sysSpecialHandler.procedureName = procedureName;
sysSpecialHandler.moduleName = moduleName;

/* Install in application table */
myErr = AEInstallSpecialHandler( keyPreDispatch, /* type of special handler
                                   (UniversalProcPtr)MyPreDispatchHandler,
                                   FALSE); /* install in application table */

/* Install in system table */
myErr = AEInstallSpecialHandler( keyPreDispatch, /* type of special handler
                                   (UniversalProcPtr)&sysSpecialHandler,
                                   TRUE); /* install in system table */
```

AEInstallSpecialHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AELaunchApplication (OS/2)

AELaunchApplication (OS/2) - Syntax

This function launches an application on a local machine.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

char      *appName;
ULONG     timeout;
PID       *thePID;
OSErr     rc;          /* Return code. */

rc = AELaunchApplication(appName, timeout,
                          thePID);
```

AELaunchApplication (OS/2) Parameter - appName

appName (char *) - input

A pointer to a buffer containing the descriptive name of the application as specified in the call to [OSAInstallApplication](#).

AELaunchApplication (OS/2) Parameter - timeout

timeout (ULONG) - input

The time-out value, in milliseconds.

AELaunchApplication (OS/2) Parameter - thePID

thePID (PID *) - output

The process ID of the created process.

AELaunchApplication (OS/2) Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

errAETimeOut

The OSA event timed out.

errAESystemError

errAppNotFound A general scripting system error occurred.
The specified application or part handler is not found in the registration data base.

AELaunchApplication (OS/2) - Parameters

appName (char *) - input
A pointer to a buffer containing the descriptive name of the application as specified in the call to [OSAInstallApplication](#).

timeOut (ULONG) - input
The time-out value, in milliseconds.

thePID (PID *) - output
The process ID of the created process.

rc (OSError) - returns
Return code.

noErr	No error.
errAETimeOut	The OSA event timed out.
errAESystemError	A general scripting system error occurred.
errAppNotFound	The specified application or part handler is not found in the registration data base.

AELaunchApplication (OS/2) - Remarks

The application is not launched if another copy is already running.

This function blocks the caller until the application has been launched and has issued an [AEInit](#) call.

The registration data base is queried to get the application's path to the executable file.

AELaunchApplication (OS/2) - Example Code

This example uses the AELaunchApplication method to start up the application "My App" and wait 60 ms for it to register itself with the OSA Event Manager.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

OSError myErr;           /* result code */
PID thePID;              /* process ID of application */
char appName[] = "My App"; /* name of application, as registered */
ULONG timeOut = 60;      /* timeout in milliseconds */

myErr = AELaunchApplication(appName, timeOut, &thePID);
```

AELaunchApplication (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEManagerInfo

AEManagerInfo - Syntax

This function obtains information about the version of the OSA Event Manager currently available or the number of processes that are currently recording OSA events.

```
#define INCL_OSAEVENTS
#include <os2.h>

AEKeyword    keyword;
long         *result;
OSErr        rc;      /* Return code. */

rc = AEManagerInfo(keyword, result);
```

AEManagerInfo Parameter - keyword

keyword ([AEKeyword](#)) - input

A value that determines what kind of information is to be returned. The value can be represented by one of these constants:

keyAERecorderCount

keyAEVersion

AEManagerInfo Parameter - result

result (long *) - input

If the value of the *keyword* parameter is keyAERecorderCount, this parameter is an integer that indicates the number of processes that are currently recording OSA event.

If the value of the *keyword* parameter is keyAEVersion, this parameter is an integer that provides information about the version of the OSA Event Manager available on the current computer, using the same format as a **vers** resource.

AEManagerInfo Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.
ERROR_INVALID_PARAMETER
An invalid keyword is specified.

AEManagerInfo - Parameters

keyword ([AEKeyword](#)) - input
A value that determines what kind of information is to be returned. The value can be represented by one of these constants:

keyAERecorderCount
keyAEVersion

result (long *) - input
If the value of the *keyword* parameter is keyAERecorderCount, this parameter is an integer that indicates the number of processes that are currently recording OSA event.

If the value of the *keyword* parameter is keyAEVersion, this parameter is an integer that provides information about the version of the OSA Event Manager available on the current computer, using the same format as a **vers** resource.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.
ERROR_INVALID_PARAMETER
An invalid keyword is specified.

AEManagerInfo - Remarks

For information about using the AEManagerInfo function to check whether OSA event recording is on or not, see [Recording OSA Events](#).

AEManagerInfo - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEProcessOSAEvent (OS/2)

AEProcessOSAEvent (OS/2) - Syntax

This function calls the appropriate handler for a specified OSA event.

```
#define INCL_OSAEVENTS
#include <os2.h>

const SemanticEvent *theSemanticEvent;
OSErr rc; /* Return code. */

rc = AEProcessOSAEvent(theSemanticEvent);
```

AEProcessOSAEvent (OS/2) Parameter - theSemanticEvent

theSemanticEvent (`const SemanticEvent *`) - input
A pointer to the semantic event.

AEProcessOSAEvent (OS/2) Return Value - rc

rc (`OSErr`) - returns
Return code.

noErr	No error occurred.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
ERROR_INSUFFICIENT_BUFFER	The size of the buffer is not large enough to hold the data to be returned.
errAECorruptData	Data in an OSA event could not be read.
errAENewerVersion	A newer version of the OSA Event Manager is needed.
errAEEventNotHandled	The event was not handled by an OSA event handler.

AEProcessOSAEvent (OS/2) - Parameters

theSemanticEvent ([const SemanticEvent *](#)) - input

A pointer to the semantic event.

rc ([OSErr](#)) - returns

Return code.

noErr

No error occurred.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

ERROR_INSUFFICIENT_BUFFER

The size of the buffer is not large enough to hold the data to be returned.

errAECorruptData

Data in an OSA event could not be read.

errAENewerVersion

A newer version of the OSA Event Manager is needed.

errAEEventNotHandled

The event was not handled by an OSA event handler.

AEProcessOSAEvent (OS/2) - Remarks

This method looks first in the application's special handler dispatch table for an entry that was installed with the constant keyPreDispatch. If the application's special handler dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, the function looks in the application's OSA event dispatch table for an entry that matches the event class and event ID of the specified OSA event.

If the application's OSA event dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, the AEProcessOSAEvent method looks in the system special handler dispatch table for an entry that was installed with the constant keyPreDispatch. If the system special handler dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, this method looks at the system OSA event dispatch table for an entry that matches the event class and event ID of the specified event.

If the system OSA event dispatch table does not include such a handler, the OSA Event Manager returns the result code errAEEventNotHandled to the server application and, if the client application is waiting for a reply, to the client application.

If the AEProcessOSAEvent method finds an entry in one of the dispatch tables that matches the event class and event ID of the OSA event, it calls the corresponding handler.

If an OSA event dispatch table contains one entry for an event class and a specified event ID, and also contains another entry that is identical except that it specifies a wild card value for either the event class or the event ID, the OSA Event Manager dispatches the more specific entry. For example, if an OSA event dispatch table includes one entry that specifies the event class as kAECoreSuite and the event ID as kAEDelete, and another entry that specifies the event class and kAECoreSuite and the event ID as typeWildcard, the OSA Event Manager dispatches the OSA event handler associated with the entry that specifies the event ID as kAEDelete.

AEProcessOSAEvent (OS/2) - Example Code

This example shows how to dispatch WM_SEMANTICEVENT messages to the AEProcessOSAEvent method.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#define INCL_WIN
#include <os2.h>
```

```
MRESULT EXPENTRY WindowProc(HWND hwnd, MSGID msg, MPARAM mp1, MPARAM mp2)
{
    switch(msg)
```

```

{
    case WM_SEMANTICEVENT:
    {
        SemanticEvent *semEvent = (SemanticEvent *)mpl;
        myErr = AEProcessOSAEvent(semEvent);
    }
    break;
    ...
}
}

```

AEProcessOSAEvent (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEPutArray

AEPutArray - Syntax

This function puts the data for an OSA event array into any descriptor list.

```

#define INCL_OSAEVENTS
#include <os2.h>

const AEDescList      *theAEDescList;
AEArrayType           arrayType;
const AEArrayData     *arrayPtr;
DescType              itemType;
Size                  itemSize;
long                  itemCount;
OSErr                 rc;          /* Return code. */

rc = AEPutArray(theAEDescList, arrayType,
                arrayPtr, itemType, itemSize, itemCount);

```

AEPutArray Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input

The descriptor list into which to put the OSA event array. If there are any items already in the descriptor list, they are replaced.

AEPutArray Parameter - arrayType

arrayType ([AEArrayType](#)) - input

The OSA event array type to be created. This is specified by one of the following constants:

kAEDataArray	An array of integers.
kAEPackedArray	An array of characters.
kAEHandleArray	An array of handles.
kAEDescArray	An array of descriptors.
kAEKeyDescArray	An array of keyword descriptors.

AEPutArray Parameter - arrayPtr

arrayPtr ([const AEArrayData *](#)) - input

A pointer to the buffer containing the array.

AEPutArray Parameter - itemType

itemType ([DescType](#)) - input

The descriptor type of array items to be created (for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray).

AEPutArray Parameter - itemSize

itemSize ([Size](#)) - input

Size, in bytes, of the array items to be created (for arrays of type kAEDataArray or kAEPackedArray).

AEPutArray Parameter - itemCount

itemCount (long) - input

The number of elements in the array.

AEPutArray Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.

AEPutArray - Parameters

theAEDescList ([const AEDescList *](#)) - input
The descriptor list into which to put the OSA event array. If there are any items already in the descriptor list, they are replaced.

arrayType ([AEArrayType](#)) - input
The OSA event array type to be created. This is specified by one of the following constants:

kAEDataArray	An array of integers.
kAEPackedArray	An array of characters.
kAEHandleArray	An array of handles.
kAEDescArray	An array of descriptors.
kAEKeyDescArray	An array of keyword descriptors.

arrayPtr ([const AEArrayData *](#)) - input
A pointer to the buffer containing the array.

itemType ([DescType](#)) - input
The descriptor type of array items to be created (for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray).

itemSize ([Size](#)) - input
Size, in bytes, of the array items to be created (for arrays of type kAEDataArray or kAEPackedArray).

itemCount (long) - input
The number of elements in the array.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.

AEPutArray - Remarks

When you use `AEPutArray` to put an array into a factored descriptor list, each array item must include the data that is common to all the descriptor records in the list. The OSA Event Manager automatically isolates the data you specified in the call to [AECreatelist](#) that is common to all the elements of the array.

For information about data types and constants used with `AEPutArray`, see [OSA Event Array Data Types](#).

AEPutArray - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEPutAttributeDesc

AEPutAttributeDesc - Syntax

This function adds a descriptor record and a keyword to an OSA event as an attribute.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
const AEDesc        *theAEDesc;
OSErr               rc;          /* Return code. */

rc = AEPutAttributeDesc(theOSAEvent, theAEKeyword,
                        theAEDesc);
```

AEPutAttributeDesc Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to which you are adding an attribute.

AEPutAttributeDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword for the attribute to be added.

If the OSA event already included an attribute with this keyword, the attribute is replaced. This parameter can be set to any of the constants shown in the following list:

keyAddressAttr	Address of target application
keyEventClassAttr	Event class
keyEventIDAttr	Event ID
keyEventSourceAttr	Source application
keyInteractLevelAttr	Settings to allow the OSA Event Manager to bring server application to the foreground
keyMissedKeywordAttr	First required parameter remaining in OSA event
keyOptionalKeywordAttr	List of optional parameters for OSA event
keyOriginalAddressAttr	Address of original source of OSA event
keyReturnIDAttr	Return ID for reply OSA event
keyTimeoutAttr	Length of time in ticks that client will wait for reply or result from the server
keyTransactionIDAttr	Transaction ID identifying a series of OSA events

AEPutAttributeDesc Parameter - theAEDesc

theAEDesc (**const AEDesc ***) - input

The descriptor record to be assigned to the attribute. The descriptor type of the specified descriptor record should match the defined descriptor type for that attribute. For example, the keyEventSourceAttr attribute has the typeShortInteger descriptor type.

AEPutAttributeDesc Return Value - rc

rc (**OSErr**) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEC coercionFail	Data could not be coerced to the requested descriptor type.
errAENotAEDesc	The descriptor record is invalid.

AEPutAttributeDesc - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to which you are adding an attribute.

theAEKeyword ([AEKeyword](#)) - input
The keyword for the attribute to be added.

If the OSA event already included an attribute with this keyword, the attribute is replaced. This parameter can be set to any of the constants shown in the following list:

keyAddressAttr	Address of target application
keyEventClassAttr	Event class
keyEventIDAttr	Event ID
keyEventSourceAttr	Source application
keyInteractLevelAttr	Settings to allow the OSA Event Manager to bring server application to the foreground
keyMissedKeywordAttr	First required parameter remaining in OSA event
keyOptionalKeywordAttr	List of optional parameters for OSA event
keyOriginalAddressAttr	Address of original source of OSA event
keyReturnIDAttr	Return ID for reply OSA event
keyTimeoutAttr	Length of time in ticks that client will wait for reply or result from the server
keyTransactionIDAttr	Transaction ID identifying a series of OSA events

theAEDesc ([const AEDesc *](#)) - input
The descriptor record to be assigned to the attribute. The descriptor type of the specified descriptor record should match the defined descriptor type for that attribute. For example, the keyEventSourceAttr attribute has the typeShortInteger descriptor type.

rc ([OSError](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAECOercionFail	Data could not be coerced to the requested descriptor type.
errAENotAEDesc	The descriptor record is invalid.

AEPutAttributeDesc - Remarks

This function takes a descriptor record and a keyword and adds them to an OSA event as an attribute. If the descriptor type required for the attribute is different from the descriptor type of the descriptor record, the OSA Event Manager attempts to coerce the descriptor record into the required type, with one exception: the OSA Event Manager does not attempt to coerce the data for an address attribute, thereby allowing applications to use their own address types.

AEPutAttributeDesc - Example Code

For an example of the use of AEPutAttributeDesc, see the figure in section [Specifying Optional Parameters for an OSA Event](#).

AEPutAttributeDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEPutAttributePtr

AEPutAttributePtr - Syntax

This function adds a pointer to data, a descriptor type, and a keyword to an OSA event as an attribute.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent    theOSAEvent;
AEKeyword         theAEKeyword;
DescType          typeCode;
const void        *dataPtr;
Size              dataSize;
OSErr             rc;          /* Return code. */

rc = AEPutAttributePtr(theOSAEvent, theAEKeyword,
                      typeCode, dataPtr, dataSize);
```

AEPutAttributePtr Parameter - theOSAEvent

theOSAEvent ([const OSAEvent](#)) - input
The OSA event to which to add an attribute.

AEPutAttributePtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword for the attribute to be added.

If the OSA event already included an attribute with this keyword, the attribute is replaced. This parameter can be any of the constants shown in the following list:

keyAddressAttr	Address of target application
keyEventClassAttr	Event class
keyEventIDAttr	Event ID
keyEventSourceAttr	Source application
keyInteractLevelAttr	Settings to allow the OSA Event Manager to bring server application to the foreground
keyMissedKeywordAttr	First required parameter remaining in OSA event
keyOptionalKeywordAttr	List of optional parameters for OSA event
keyOriginalAddressAttr	Address of original source of OSA event
keyReturnIDAttr	Return ID for reply OSA event
keyTimeoutAttr	Length of time in ticks that client will wait for reply or result from the server
keyTransactionIDAttr	Transaction ID identifying a series of OSA events

AEPutAttributePtr Parameter - typeCode

typeCode ([DescType](#)) - input

The descriptor type for the attribute.

AEPutAttributePtr Parameter - dataPtr

dataPtr (const void *) - input

A pointer to the buffer containing the data to be assigned to the attribute.

AEPutAttributePtr Parameter - dataSize

dataSize ([Size](#)) - input

The length, in bytes, of the data to be assigned to the attribute.

AEPutAttributePtr Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEC coercionFail	Data could not be coerced to the requested descriptor type.
errAENotAEDesc	The descriptor record is invalid.

AEPutAttributePtr - Parameters

theOSAEvent ([const OSAEvent](#)) - input
The OSA event to which to add an attribute.

theAEKeyword ([AEKeyword](#)) - input
The keyword for the attribute to be added.

If the OSA event already included an attribute with this keyword, the attribute is replaced. This parameter can be any of the constants shown in the following list:

keyAddressAttr	Address of target application
keyEventClassAttr	Event class
keyEventIDAttr	Event ID
keyEventSourceAttr	Source application
keyInteractLevelAttr	Settings to allow the OSA Event Manager to bring server application to the foreground
keyMissedKeywordAttr	First required parameter remaining in OSA event
keyOptionalKeywordAttr	List of optional parameters for OSA event
keyOriginalAddressAttr	Address of original source of OSA event
keyReturnIDAttr	Return ID for reply OSA event
keyTimeoutAttr	Length of time in ticks that client will wait for reply or result from the server
keyTransactionIDAttr	Transaction ID identifying a series of OSA events

typeCode ([DescType](#)) - input
The descriptor type for the attribute.

dataPtr ([const void *](#)) - input
A pointer to the buffer containing the data to be assigned to the attribute.

dataSize ([Size](#)) - input

The length, in bytes, of the data to be assigned to the attribute.

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEC coercionFail

Data could not be coerced to the requested descriptor type.

errAENotAEDesc

The descriptor record is invalid.

AEPutAttributePtr - Remarks

This function adds the specified pointer to data, descriptor type, and keyword to the specified OSA event as an attribute.

AEPutAttributePtr - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AEPutDesc

AEPutDesc - Syntax

This function adds a descriptor record to any descriptor list.

```
#define INCL_OSAEVENTS
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
const AEDesc        *theAEDesc;
OSError             rc;           /* Return code. */

rc = AEPutDesc(theAEDescList, index, theAEDesc);
```

AEPutDesc Parameter - theAEDescList

theAEDescList (`const AEDescList *`) - input
The descriptor list to which to add a descriptor record.

AEPutDesc Parameter - index

index (`long`) - input
The position of the descriptor record in the descriptor list. (For example, the value 2 specifies the second descriptor record in the list.) If there is already a descriptor record in the specified position, it is replaced. If the value of index is 0, the descriptor record is added to the end of the list.

AEPutDesc Parameter - theAEDesc

theAEDesc (`const AEDesc *`) - input
The descriptor record to be added to the list.

AEPutDesc Return Value - rc

rc (`OSErr`) - returns
Return code.

<code>noErr</code>	No error.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	There is not enough memory.
<code>errAEWrongDataType</code>	Wrong descriptor type.
<code>errAEBadListItem</code>	An operation involving a list item failed.
<code>errAEIllegalIndex</code>	The list index is invalid.

AEPutDesc - Parameters

theAEDescList (`const AEDescList *`) - input
The descriptor list to which to add a descriptor record.

index (`long`) - input
The position of the descriptor record in the descriptor list. (For example, the value 2 specifies the second descriptor record in the list.) If there is already a descriptor record in the specified position, it is replaced. If the value of index is 0, the descriptor record is added to the end of the list.

theAEDesc (`const AEDesc *`) - input
The descriptor record to be added to the list.

rc ([OSError](#)) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAETWrongDataType	Wrong descriptor type.
errAETBadListItem	An operation involving a list item failed.
errAETIllegalIndex	The list index is invalid.

AEPutDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AEPutKeyDesc

AEPutKeyDesc - Syntax

This function adds a descriptor record and a keyword to an AE record as a keyword-specified descriptor record.

```
#define INCL_OSAEVENTS
#include <os2.h>

AERecord    *theAERecord;
AEKeyword   theAEKeyword;
AEDesc      *theAEDesc;
OSError      rc;          /* Return code. */

rc = AEPutKeyDesc(theAERecord, theAEKeyword,
                  theAEDesc);
```

AEPutKeyDesc Parameter - theAERecord

theAERecord ([AERecord *](#)) - input
The AE record to which to add the keyword-specified descriptor record.

AEPutKeyDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword specifying the descriptor record. If there was already a keyword-specified descriptor record with this keyword, it is replaced.

AEPutKeyDesc Parameter - theAEDesc

theAEDesc ([AEDesc](#) *) - input

The descriptor record for the keyword-specified descriptor record.

AEPutKeyDesc Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEWrongDataType

A wrong descriptor type is specified, or the keyword is set as the wildcard.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEPutKeyDesc - Parameters

theAERecord ([AERecord](#) *) - input

The AE record to which to add the keyword-specified descriptor record.

theAEKeyword ([AEKeyword](#)) - input

The keyword specifying the descriptor record. If there was already a keyword-specified descriptor record with this keyword, it is replaced.

theAEDesc ([AEDesc](#) *) - input

The descriptor record for the keyword-specified descriptor record.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEWrongDataType

A wrong descriptor type is specified, or the keyword is set as the wildcard.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEPutKeyDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AEPutKeyPtr

AEPutKeyPtr - Syntax

This function adds a pointer to data, a descriptor type, and a keyword to an AE record as a keyword-specified descriptor record.

```
#define INCL_OSAEVENTS
#include <os2.h>

AERecord      *theAERecord;
AEKeyword      theAEKeyword;
DescType      typeCode;
void          *dataPtr;
Size          dataSize;
OSErr         rc;          /* Return code. */

rc = AEPutKeyPtr(theAERecord, theAEKeyword,
                 typeCode, dataPtr, dataSize);
```

AEPutKeyPtr Parameter - theAERecord

theAERecord ([AERecord](#) *) - input
The AE record to which to add a keyword-specified descriptor record.

AEPutKeyPtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that identifies the descriptor record. If the AE record already includes a descriptor record with this keyword, it is replaced.

AEPutKeyPtr Parameter - typeCode

typeCode ([DescType](#)) - input
The descriptor type for the keyword-specified descriptor record.

AEPutKeyPtr Parameter - dataPtr

dataPtr - input
A pointer to the data for the keyword-specified descriptor record.

AEPutKeyPtr Parameter - dataSize

dataSize ([Size](#)) - input
The length, in bytes, of the data for the keyword-specified descriptor record.

AEPutKeyPtr Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
memFullErr	There is not enough memory.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.

AEPutKeyPtr - Parameters

theAERecord ([AERecord *](#)) - input
The AE record to which to add a keyword-specified descriptor record.

theAEKeyword ([AEKeyword](#)) - input
The keyword that identifies the descriptor record. If the AE record already includes a descriptor record with this keyword, it is replaced.

typeCode ([DescType](#)) - input
The descriptor type for the keyword-specified descriptor record.

dataPtr - input
A pointer to the data for the keyword-specified descriptor record.

dataSize ([Size](#)) - input

The length, in bytes, of the data for the keyword-specified descriptor record.

rc ([OSErr](#)) - returns
Return code.

<code>noErr</code>	No error.
<code>memFullErr</code>	There is not enough memory.
<code>errAEWrongDataType</code>	A wrong descriptor type is specified.
<code>errAENotAEDesc</code>	The descriptor record is invalid.
<code>errAEBadListItem</code>	An operation involving a list item failed.

AEPutKeyPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AEPutParamDesc

AEPutParamDesc - Syntax

This function adds a descriptor record and a keyword to an OSA event as an OSA event parameter.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
AEDesc              *theAEDesc;
OSErr               rc;          /* Return code. */

rc = AEPutParamDesc(theOSAEvent, theAEKeyword,
                    theAEDesc);
```

AEPutParamDesc Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to which to add a parameter.

AEPutParamDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword for the parameter to be added. If the OSA event already included a parameter with this keyword, the parameter is replaced.

AEPutParamDesc Parameter - theAEDesc

theAEDesc ([AEDesc *](#)) - input

The descriptor record for the parameter.

AEPutParamDesc Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEWrongDataType

A wrong descriptor type is specified.

errAENotAEDesc

The descriptor record is invalid.

errAEBadListItem

An operation involving a list item failed.

AEPutParamDesc - Parameters

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event to which to add a parameter.

theAEKeyword ([AEKeyword](#)) - input

The keyword for the parameter to be added. If the OSA event already included a parameter with this keyword, the parameter is replaced.

theAEDesc ([AEDesc *](#)) - input

The descriptor record for the parameter.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEWrongDataType

A wrong descriptor type is specified.

errAENotAEDesc

errAEBadListItem The descriptor record is invalid.
 An operation involving a list item failed.

AEPutParamDesc - Example Code

For an example of the use of AEPutParamDesc, see [Adding Parameters to an OSA Event](#).

AEPutParamDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AEPutParamPtr

AEPutParamPtr - Syntax

This function adds a pointer to data, a descriptor type, and a keyword to an OSA event as an OSA event parameter.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *theOSAEvent;
AEKeyword           theAEKeyword;
DescType            typeCode;
const void          *dataPtr;
Size                dataSize;
OSErr               rc;          /* Return code. */

rc = AEPutParamPtr(theOSAEvent, theAEKeyword,
                  typeCode, dataPtr, dataSize);
```

AEPutParamPtr Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to which to add a parameter.

AEPutParamPtr Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input

The keyword for the parameter to be added. If the OSA event already included a parameter with this keyword, the parameter is replaced.

AEPutParamPtr Parameter - typeCode

typeCode ([DescType](#)) - input

The descriptor type for the parameter.

AEPutParamPtr Parameter - dataPtr

dataPtr (const void *) - input

A pointer to the data for the parameter.

AEPutParamPtr Parameter - dataSize

dataSize ([Size](#)) - input

The length, in bytes, of the data for the parameter.

AEPutParamPtr Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr	No error.
memFullErr	There is not enough memory.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is not valid.
errAEBadListItem	An operation involving a list item failed.

AEPutParamPtr - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to which to add a parameter.

theAEKeyword ([AEKeyword](#)) - input
The keyword for the parameter to be added. If the OSA event already included a parameter with this keyword, the parameter is replaced.

typeCode ([DescType](#)) - input
The descriptor type for the parameter.

dataPtr ([const void *](#)) - input
A pointer to the data for the parameter.

dataSize ([Size](#)) - input
The length, in bytes, of the data for the parameter.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
memFullErr	There is not enough memory.
errAEWrongDataType	A wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is not valid.
errAEBadListItem	An operation involving a list item failed.

AEPutParamPtr - Example Code

For an example of the use of AEPutParamPtr, see [Adding Parameters to an OSA Event](#).

AEPutParamPtr - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AEPutPtr

AEPutPtr - Syntax

This function adds data specified in a buffer to any descriptor list as a descriptor record.

```

#define INCL_OSAEVENTS
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
DescType            typeCode;
const void          *dataPtr;
Size                dataSize;
OSErr               rc;                /* Return code. */

rc = AEPutPtr(theAEDescList, index, typeCode,
              dataPtr, dataSize);

```

AEPutPtr Parameter - theAEDescList

theAEDescList ([const AEDescList *](#)) - input
 The descriptor list to which to add a descriptor record.

AEPutPtr Parameter - index

index (long) - input
 The position of the descriptor record in the descriptor list. (For example, the value 2 specifies the second descriptor record in the list.) If there is already a descriptor record in the specified position, it is replaced. If the value of index is 0, the descriptor record is added to the end of the list.

AEPutPtr Parameter - typeCode

typeCode ([DescType](#)) - input
 The descriptor type for the resulting descriptor record.

AEPutPtr Parameter - dataPtr

dataPtr (const void *) - input
 A pointer to the data for the descriptor record.

AEPutPtr Parameter - dataSize

dataSize ([Size](#)) - input

The length, in bytes, of the data for the descriptor record.

AEPutPtr Return Value - rc

rc ([OSError](#)) - returns
Return code.

noErr	No error.
memFullErr	There is not enough memory.
errAETWrongDataType	The wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.
errAEIllegalIndex	The list index is invalid.

AEPutPtr - Parameters

theAEDescList ([const AEDescList *](#)) - input
The descriptor list to which to add a descriptor record.

index (long) - input
The position of the descriptor record in the descriptor list. (For example, the value 2 specifies the second descriptor record in the list.) If there is already a descriptor record in the specified position, it is replaced. If the value of index is 0, the descriptor record is added to the end of the list.

typeCode ([DescType](#)) - input
The descriptor type for the resulting descriptor record.

dataPtr (const void *) - input
A pointer to the data for the descriptor record.

dataSize ([Size](#)) - input
The length, in bytes, of the data for the descriptor record.

rc ([OSError](#)) - returns
Return code.

noErr	No error.
memFullErr	There is not enough memory.
errAETWrongDataType	The wrong descriptor type is specified.
errAENotAEDesc	The descriptor record is invalid.
errAEBadListItem	An operation involving a list item failed.
errAEIllegalIndex	The list index is invalid.

AEPutPtr - Example Code

For an example of the use of AEPutPtr, see the figure in section [Specifying Optional Parameters for an OSA Event](#).

AEPutPtr - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AERemoveCoercionHandler (OS/2)

AERemoveCoercionHandler (OS/2) - Syntax

This function removes a coercion handler from the application's coercion handler dispatch table.

```
#define INCL_OSAEVENTS
#include <os2.h>

DescType      fromType;
DescType      toType;
AERemoveCoercionHandlerUPP handler;
BOOL          isSysHandler;
OS2Err        rc; /* Return code. */

rc = AERemoveCoercionHandler(fromType, toType,
                             handler, isSysHandler);
```

AERemoveCoercionHandler (OS/2) Parameter - fromType

fromType ([DescType](#)) - input
The descriptor type of the data coerced by the handler.

AERemoveCoercionHandler (OS/2) Parameter - toType

toType ([DescType](#)) - input
The descriptor type of the resulting data.

AERemoveCoercionHandler (OS/2) Parameter - handler

handler ([AECoectionHandlerUPP](#)) - input

A pointer to the coercion handler. Although the *fromType* and *toType* parameters are sufficient to identify the handler to be removed, providing the *handler* parameter is a safeguard that ensures the correct handler is removed.

The *handler* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

AERemoveCoercionHandler (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system coercion table.

TRUE

Handler is removed from the system coercion table.

FALSE

Handler is removed from the application's coercion table.

AERemoveCoercionHandler (OS/2) Return Value - rc

rc ([OSError](#)) - returns

Return code.

noErr

No error.

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

errAEHandlerNotFound

No coercion handler was found.

AERemoveCoercionHandler (OS/2) - Parameters

fromType ([DescType](#)) - input

The descriptor type of the data coerced by the handler.

toType ([DescType](#)) - input

The descriptor type of the resulting data.

handler ([AECoectionHandlerUPP](#)) - input

A pointer to the coercion handler. Although the *fromType* and *toType* parameters are sufficient to identify the handler to be removed, providing the *handler* parameter is a safeguard that ensures the correct handler is removed.

The *handler* parameter is passed as a pointer to an [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system coercion table.

TRUE	Handler is removed from the system coercion table.
FALSE	Handler is removed from the application's coercion table.

rc (**OSErr**) - returns
Return code.

noErr	No error.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory.
errAEHandlerNotFound	No coercion handler was found.

AERemoveCoercionHandler (OS/2) - Example Code

This example shows how to remove the application and system coercion handlers that coerce a date in long format into the application-defined date string.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

/* Application-defined descriptor type */
#define typeMyDateString    0x65746164    /* "date" */

/* Prototype for application coercion handler */
OSErr APIENTRY MyApplicationCoercionHandler(DescType typeCode,
                                             const void *dataPtr,
                                             Size dataSize,
                                             DescType toType,
                                             long handlerRefcon,
                                             AEDesc *result);

OSErr myErr;
AESystemHandler sysCoercionHandler;    /* system handler structure
char procedureName[] = "MySystemCoercionHandler"; /* name of coercion handle
char moduleName[]    = "mymodule.dll";    /* module where the handler resides

sysCoercionHandler.procedureName = procedureName;
sysCoercionHandler.moduleName = moduleName;

/* Remove from application table */
myErr = AERemoveCoercionHandler(typeLongDateTime,    /* input type */
                                typeMyDateString,    /* result type */
                                (AECOercionHandlerUPP)MyApplicationCoe rcic
                                FALSE);    /* remove from application tabl

/* Remove from system table */
myErr = AERemoveCoercionHandler(typeLongDateTime,    /* input type */
                                typeMyDateString,    /* result type */
                                (AECOercionHandlerUPP)&sysCoercionHandler,
                                TRUE);    /* remove from system table */
```

AERemoveCoercionHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

AERemoveEventHandler (OS/2)

AERemoveEventHandler (OS/2) - Syntax

This function removes an entry from the application's OSA event dispatch table.

```
#define INCL_OSAEVENTS
#include <os2.h>

AEEEventClass      theAEEEventClass;
AEEEventID         theAEEEventID;
AEEEventHandlerUPP handler;
BOOL               isSysHandler;
OSErr              rc;          /* Return code. */

rc = AERemoveEventHandler(theAEEEventClass,
                          theAEEEventID, handler, isSysHandler);
```

AERemoveEventHandler (OS/2) Parameter - theAEEEventClass

theAEEEventClass ([AEEEventClass](#)) - input

The event class for the handler whose entry is to be removed from the dispatch table.

AERemoveEventHandler (OS/2) Parameter - theAEEEventID

theAEEEventID ([AEEEventID](#)) - input

The event ID for the handler whose entry is to be removed from the OSA event dispatch table.

AERemoveEventHandler (OS/2) Parameter - handler

handler ([AEEEventHandlerUPP](#)) - input

A pointer to the handler to be removed. Although the *theAEEEventClass* and the *theAEEEventID* parameter are sufficient to identify the handler to be removed, providing the handler parameter is a safeguard that ensures the correct handler is removed. If the value of this parameter is NIL, the OSA Event Manager relies solely on the event class and event ID to identify the handler.

The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handler is being installed; therefore, a cast must be used to pass this parameter.

AERemoveEventHandler (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system OSA event dispatch table.

TRUE

Handler is removed from the system OSA event dispatch table.

FALSE

Handler is removed from the application's OSA event dispatch table.

AERemoveEventHandler (OS/2) Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errAEHandlerNotFound

No handler was found for the OSA event.

AERemoveEventHandler (OS/2) - Parameters

theAEEEventClass ([AEEEventClass](#)) - input

The event class for the handler whose entry is to be removed from the dispatch table.

theAEEEventID ([AEEEventID](#)) - input

The event ID for the handler whose entry is to be removed from the OSA event dispatch table.

handler ([AEEEventHandlerUPP](#)) - input

A pointer to the handler to be removed. Although the *theAEEEventClass* and the *theAEEEventID* parameter are sufficient to identify the handler to be removed, providing the handler parameter is a safeguard that ensures the correct handler is removed. If the value of this parameter is NIL, the OSA Event Manager relies solely on the event class and event ID to identify the handler.

The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handler is being installed; therefore, a cast must be used to pass this parameter.

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system OSA event dispatch table.

TRUE

Handler is removed from the system OSA event dispatch table.

FALSE

Handler is removed from the application's OSA event dispatch table.

rc (**OSErr**) - returns
Return code.

noErr

No error.

errAEHandlerNotFound

No handler was found for the OSA event.

AERemoveEventHandler (OS/2) - Remarks

This method removes the OSA event dispatch table entry specified by the *theAEEEventClass*, *theAEEEventID*, and the *handler* parameters. The *typeWildcard* constant can be specified for the *theAEEEventClass* or the *theAEEEventID* parameter or for both parameters; however, the *AERemoveEventHandler* method returns an error unless an entry exists that specifies *typeWildcard* exactly the same way. For example, if you specify *typeWildcard* in both the *theAEEEventClass* and the *theAEEEventID* parameters, the OSA Event Manager does not remove the first handler for any event class and ID in the dispatch table; instead, the dispatch table must contain an entry that specifies *typeWildcard* for both the event class and ID.

AERemoveEventHandler (OS/2) - Example Code

This example shows how to remove, from both the application and system tables, the event handlers that were installed to handle the Open Document event.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>
```

```
/* Prototype for application event handler */
OSErr APIENTRY MyOpenDocAppHandler(const OSAEvent *theOSAEvent,
                                   OSAEvent *reply,
                                   long handlerRefcon);
```

```
OSErr myErr;
AESystemHandler sysEventHandler; /* system handler structure
char procedureName[] = "MyOpenDocSysHandler"; /* name of event handler
char moduleName[] = "mymodule.dll"; /* module where the handler resides */
```

```
sysEventHandler.procedureName = procedureName;
sysEventHandler.moduleName = moduleName;
```

```
/* Remove from application table */
myErr = AERemoveEventHandler( kCoreEventClass, /* event class */
                             kAEOpenDocuments, /* event ID */
                             (AEEEventHandlerUPP)MyOpenDocAppHandler,
                             FALSE); /* remove from application table
```

```
/* Remove from system table */
myErr = AERemoveEventHandler( kCoreEventClass, /* event class */
                             kAEOpenDocuments, /* event ID */
                             (AEEEventHandlerUPP)&sysEventHandler,
                             TRUE); /* remove from system table */
```

AERemoveEventHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AERemoveSpecialHandler (OS/2)

AERemoveSpecialHandler (OS/2) - Syntax

This function removes a handler from the application's special handler dispatch table.

```
#define INCL_OSAEVENTS
#include <os2.h>

AEKeyword      functionClass;
UniversalProcPtr handler;
BOOL           isSysHandler;
OSErr          rc;          /* Return code. */

rc = AERemoveSpecialHandler(functionClass,
                             handler, isSysHandler);
```

AERemoveSpecialHandler (OS/2) Parameter - functionClass

functionClass ([AEKeyword](#)) - input

The keyword for the special handler to be removed. This parameter can be set to any of the following constants:

keyPreDispatch	Handler with the same parameters as an OSA event handler that is called immediately before the OSA Event Manager dispatches an OSA event.
keySelectProc	Handler to disable the Object Support Library (OSL).
keyAECCountProc	Object-counting function.
keyAECompareProc	Object-comparison function.
keyDisposeTokenProc	Token disposal function.
keyAEGetErrDescProc	Error callback function.

keyAEMarkTokenProc
Mark token function.

keyAEAdjustMarkProc
Mark-adjusting function.

AERemoveSpecialHandler (OS/2) Parameter - handler

handler ([UniversalProcPtr](#)) - input

A pointer to the special handler to be removed. Although the *functionClass* parameter is sufficient to identify the handler to be removed, providing the handler parameter is a safeguard that ensures the correct handler is removed.

The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

AERemoveSpecialHandler (OS/2) Parameter - isSysHandler

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system special handler dispatch table.

TRUE

The handler is removed from the system special handler dispatch table.

FALSE

The handler is removed from the application's special handler dispatch table.

AERemoveSpecialHandler (OS/2) Return Value - rc

rc ([OSError](#)) - returns

Return code.

noErr

No error.

memFullErr

There is not enough memory.

errAENotASpecialFunction

A wrong keyword for a special function was used.

AERemoveSpecialHandler (OS/2) - Parameters

functionClass ([AEKeyword](#)) - input

The keyword for the special handler to be removed. This parameter can be set to any of the following constants:

keyPreDispatch

Handler with the same parameters as an OSA event handler that is called immediately before the OSA Event Manager dispatches an OSA event.

keySelectProc

Handler to disable the Object Support Library (OSL).

keyAECCountProc
Object-counting function.

keyAECCompareProc
Object-comparison function.

keyDisposeTokenProc
Token disposal function.

keyAEGetErrDescProc
Error callback function.

keyAEMarkTokenProc
Mark token function.

keyAEAdjustMarkProc
Mark-adjusting function.

handler ([UniversalProcPtr](#)) - input

A pointer to the special handler to be removed. Although the *functionClass* parameter is sufficient to identify the handler to be removed, providing the handler parameter is a safeguard that ensures the correct handler is removed.

The *handler* parameter is passed as a pointer to a [AESystemHandler](#) structure instead of a handler entry point when a system handle is being installed; therefore, a cast must be used to pass this parameter.

isSysHandler ([BOOL](#)) - input

A flag indicating whether to remove the handler from the system special handler dispatch table.

TRUE
The handler is removed from the system special handler dispatch table.

FALSE
The handler is removed from the application's special handler dispatch table.

rc ([OSErr](#)) - returns

Return code.

noErr
No error.

memFullErr
There is not enough memory.

errAENotASpecialFunction
A wrong keyword for a special function was used.

AERemoveSpecialHandler (OS/2) - Remarks

In addition to using the AERemoveSpecialHandler method to remove specific special handlers, the method can be used to disable, within the application only, all OSA Event Manager routines that support OSA event objects-that is, all routines available to the application as a result of linking the Object Support Library (OSL) and calling the AEOBJECTINIT function.

An application that expects its copy of the OSL to move after it is installed (for example, an application that keeps it in a stand-alone code resource) needs to do this. When an application calls AEOBJECTINIT to initialize the OSL, the OSL installs the address of its routines as extensions to the pack. If those routines move, the addresses become invalid.

To disable the OSL, pass the keyword keySelectProc in the *functionClass* parameter, NIL in the *handler* parameter and FALSE in the *isSysHandler* parameter. Once you have called the AERemoveSpecialHandler method with these parameters, subsequent calls by your application to any of the OSA Event Manager routines that support OSA event objects will return errors. To initialize the OSL after disabling it with the AERemoveSpecialHandler method, the application must call AEOBJECTINIT again.

If you expect to initialize the OSL and disable it several times, call the [AERemoveObjectAccessor](#) function to remove the application's object accessor methods from the application's object accessor dispatch table before calling the AERemoveSpecialHandler method.

AERemoveSpecialHandler (OS/2) - Example Code

This example shows how to remove the special handlers installed in the application and system tables to do predispatch of events.

```
#define INCL_OSAOSL
#define INCL_OSAAPI
#include <os2.h>

/* Prototype for application predispatch handler */
OS_ERR APIENTRY MyPreDispatchHandler(const OSAEvent *theOSAEvent,
                                     OSAEvent *reply,
                                     long handlerRefcon);

OS_ERR myErr;
AESystemHandler sysSpecialHandler; /* system handler structure */
char procedureName[] = "MyPreDispatchSysHandler"; /* name of special handler */
char moduleName[] = "mymodule.dll"; /* module where the handler resides */

sysSpecialHandler.procedureName = procedureName;
sysSpecialHandler.moduleName = moduleName;

/* Remove from application table */
myErr = AERemoveSpecialHandler( keyPreDispatch, /* type of special handler */
                               (UniversalProcPtr)MyPreDispatchHandler,
                               FALSE); /* remove from application table */

/* Remove from system table */
myErr = AERemoveSpecialHandler( keyPreDispatch, /* type of special handler */
                               (UniversalProcPtr)&sysSpecialHandler,
                               TRUE); /* remove from system table */
```

AERemoveSpecialHandler (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AEResetTimer

AEResetTimer - Syntax

This function resets the time-out value for an OSA event to its starting value.

```

#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *reply;
OSErr               rc;      /* Return code. */

rc = AEResetTimer(reply);

```

AEResetTimer Parameter - reply

reply (`const OSAEvent *`) - output
 The default reply for an OSA event, provided by the OSA Event Manager.

AEResetTimer Return Value - rc

rc (`OSErr`) - returns
 Return code.

noErr	No error.
errAERReplyNotValid	AEResetTimer was passed an invalid reply.
errAESystemError	An error occurred while creating or sending a reset timer message.

AEResetTimer - Parameters

reply (`const OSAEvent *`) - output
 The default reply for an OSA event, provided by the OSA Event Manager.

rc (`OSErr`) - returns
 Return code.

noErr	No error.
errAERReplyNotValid	AEResetTimer was passed an invalid reply.
errAESystemError	An error occurred while creating or sending a reset timer message.

AEResetTimer - Remarks

When your application calls AEResetTimer, the OSA Event Manager for the server application uses the default reply to send a Reset Timer event to the client application; the OSA Event Manager for the client application's computer intercepts this OSA event and resets the client application's timer for the OSA event. (The Reset Timer event is never dispatched to a handler, so the client application does not need a handler for it.)

AEResetTimer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AEResumeTheCurrentEvent

AEResumeTheCurrentEvent - Syntax

This function informs the OSA Event Manager that your application wants to resume the handling of a previously suspended OSA event or that it has completed the handling of the OSA event.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *theOSAEvent;
const OSAEvent      *reply;
AEEventHandlerUPP   dispatcher;
long                handlerRefcon;
OSErr               rc;

rc = AEResumeTheCurrentEvent(theOSAEvent,
                             reply, dispatcher, handlerRefcon);
```

AEResumeTheCurrentEvent Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to be resumed.

AEResumeTheCurrentEvent Parameter - reply

reply ([const OSAEvent *](#)) - input
The default reply provided by the OSA Event Manager for the OSA event.

AEResumeTheCurrentEvent Parameter - dispatcher

dispatcher ([AEventHandlerUPP](#)) - input
A pointer to a routine for handling the event.

This parameter can also be set to one of the following constants:

kAENoDispatch
Tells the OSA Event Manager that the OSA event has been completely processed and need not be dispatched.

kAEUseStandardDispatch
Tells the OSA Event Manager to dispatch the resumed event using the standard dispatching scheme it uses for other OSA events.

AEResumeTheCurrentEvent Parameter - handlerRefcon

handlerRefcon (long) - input
If the value of the dispatcher parameter is not **kAEUseStandardDispatch**, this parameter is the reference constant passed to the handler when the handler is called. If the value of the dispatcher parameter is **kAEUseStandardDispatch**, the OSA Event Manager ignores this parameter and instead passes the reference constant stored in the OSA event dispatch table entry for the OSA event. (You may wish to pass the same reference constant that is stored in the OSA event dispatch table. If so, call the [AEventHandlerUPP](#) function.) In addition to the following return codes,

AEResumeTheCurrentEvent Return Value - rc

rc ([OSError](#)) - returns
Return codes. In addition to the following return codes, **AEResumeTheCurrentEvent** also returns any other result returned by the event handler that was called, if any.

noErr
No error.

errAENotAEDesc
The descriptor record was invalid.

ERROR_INVALID_PARAMETER
An invalid descriptor type for the OSA event was specified.

AEResumeTheCurrentEvent - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to be resumed.

reply ([const OSAEvent *](#)) - input
The default reply provided by the OSA Event Manager for the OSA event.

dispatcher ([AEventHandlerUPP](#)) - input
A pointer to a routine for handling the event.

This parameter can also be set to one of the following constants:

kAENoDispatch
Tells the OSA Event Manager that the OSA event has been completely processed and need not be dispatched.

kAEUseStandardDispatch
Tells the OSA Event Manager to dispatch the resumed event using the standard dispatching scheme it uses for

other OSA events.

handlerRefcon (long) - input

If the value of the dispatcher parameter is not `kAEUseStandardDispatch`, this parameter is the reference constant passed to the handler when the handler is called. If the value of the dispatcher parameter is `kAEUseStandardDispatch`, the OSA Event Manager ignores this parameter and instead passes the reference constant stored in the OSA event dispatch table entry for the OSA event. (You may wish to pass the same reference constant that is stored in the OSA event dispatch table. If so, call the [AEGetEventHandler](#) function.) In addition to the following return codes,

rc ([OSErr](#)) - returns

Return codes. In addition to the following return codes, `AEResumeTheCurrentEvent` also returns any other result returned by the event handler that was called, if any.

`noErr`

No error.

`errAENotAEDesc`

The descriptor record was invalid.

`ERROR_INVALID_PARAMETER`

An invalid descriptor type for the OSA event was specified.

AEResumeTheCurrentEvent - Remarks

When your application calls the `AEResumeTheCurrentEvent` function, the OSA Event Manager resumes handling the specified OSA event using the handler specified in the dispatcher parameter, if any. If `kAENoDispatch` is specified in the dispatcher parameter, `AEResumeTheCurrentEvent` simply informs the OSA Event Manager that the specified event has been handled.

Special Considerations

An OSA event handler that suspends an event should not immediately call `AEResumeTheCurrentEvent`, or else the handler will generate an error. Instead, the handler should return just after suspending the event. When your application calls `AEResumeTheCurrentEvent` for an event that was not directly dispatched, the OSA Event Manager disposes of the event and the reply, just as it normally does, after the event handler returns to [AEProcessOSAEvent](#). Make sure all processing involving the event or the reply has been completed before your application calls `AEResumeTheCurrentEvent`. Do not call `AEResumeTheCurrentEvent` for an event that was not suspended. When your application calls `AEResumeTheCurrentEvent` for an event that was directly dispatched, your application is responsible for disposing of the original event and the reply, because it acts as both the server and the client.

AEResumeTheCurrentEvent - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AESEND (OS/2)

AESEND (OS/2) - Syntax

This function sends an OSA event.

```

#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent      *theOSAEvent;
OSAEvent            *reply;
AESendMode          sendMode;
AESendPriority       sendPriority;
long                timeOutInTicks;
AEIdleUPP           idleProc;
AEFilterUPP         filterProc;
OSErr               rc;          /* Return code. */

rc = AESend(theOSAEvent, reply, sendMode,
            sendPriority, timeOutInTicks, idleProc,
            filterProc);

```

AESend (OS/2) Parameter - theOSAEvent

theOSAEvent (`const OSAEvent *`) - input
The OSA event to be sent.

AESend (OS/2) Parameter - reply

reply (`OSAEvent *`) - output
The reply OSA event. This parameter is returned by the AESend method when `kAEWaitReply` flag is specified in the `sendMode` parameter. If the `kAEQueueReply` flag is specified in the `sendMode` parameter and the event is not going to be directly dispatched, the reply OSA event is received in the event queue. If `kAENoReply` flag is specified, the reply OSA event returned by this function is a null-descriptor record. If `kAEWaitReply` is specified in the `sendMode` parameter, the application is responsible for using the `AEDisposeDesc` method to dispose of the descriptor record returned in the reply parameter.

AESend (OS/2) Parameter - sendMode

sendMode (`AESendMode`) - input
A flag indicating the appropriate mode.

These mode flags are divided into five categories. The `kAEAlwaysInteract`, `kAECanInteract`, and `kAENeverInteract` flags are used to communicate the user interaction preference to the server application, and only one of these can be set. The `kAECanSwitchLayers` flag is used to set the application switch mode. The flags `kAENoReply`, `kAEQueueReply`, and `kAEWaitReply` specify the reply mode for an OSA event, and only one of these can be set. The last flag, `kAEWantReceipt`, sets the return receipt mode.

If your application is sending an event to itself, you can set the `kAEDontRecord` or the `kAEDontExecute` flag to prevent the event from being recorded or to ask the OSA Event Manager to record the event without your application actually receiving it.

The server is responsible for interrogating the following flags and honoring the client's requirements: `kAENeverInteract`, `kAECannotInteract`, `kAEAlwaysInteract`, and `kAECanSwitchLayers`.

`kAEAlwaysInteract`

The server application can interact with the user in response to the OSA event-by convention, whenever the server application normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user.

`kAECanInteract`

The server application can interact with the user in response to the OSA event-by convention, if the user needs to

supply information to the server. This flag is the default when an OSA event is sent to a local application.

kAEDontExecute

Your application is sending an OSA event to itself for recording purposes only-that is, you want the OSA Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

kAEDontRecord

Your application is sending an event to itself but does not want the event recorded. When OSA event recording is on, the OSA Event Manager records a copy of every event your application sends to itself except for those events for which this flag is set.

kAENeverInteract

The server application should never interact with the user in response to the OSA event.

kAENoReply

Your application does not want a reply OSA event; the server processes your OSA event as soon as it has the opportunity. The OSA Event Manager returns immediately from the AESend function.

kAEQueueReply

Your application wants a reply OSA event; the reply appears in your event queue as soon as the server has the opportunity to process and respond to your OSA event. The OSA Event Manager returns immediately from the AESend function, except for direct dispatched events. Direct dispatched events cause the reply to be returned in the *reply* parameter of this function.

kAEWaitReply

Your application wants a reply OSA event and is willing to give up the processor while waiting for the reply; for example, if the server application is on the same computer as your application, your application yields the processor to allow the server to respond to your OSA event. The OSA Event Manager creates a new thread for the calling process and blocks on this thread, waiting for the server to reply.

The process will continue to receive all messages (events) on its message queue.

kAEWantReceipt

The sender wants to receive a return receipt for this OSA event from the OSA Event Manager. (A return receipt means only that the receiving application accepted the OSA event; the OSA event may or may not be handled successfully after it is accepted.) If the receiving application does not send a returned receipt before the request times out, AESend returns errAETimeout as its function result.

AESend (OS/2) Parameter - sendPriority

sendPriority ([AESendPriority](#)) - input

A flag indicating the send priority. This parameter can be set to one of the following values:

kAENormalPriority

OSA event is posted at the end of the event queue.

kAEHighPriority

OSA event is posted at the front of the event queue.

AESend (OS/2) Parameter - timeOutInTicks

timeOutInTicks (long) - input

The length of time, in ticks, that the client application is willing to wait for the reply or return receipt from the server application before timing out. This parameter can also be set to one of the constants in the following list:

kAEDefaultTimeout

The OSA Event Manager provides an appropriate time-out duration.

kAENoTimeout

The OSA event never times out.

AESend (OS/2) Parameter - idleProc

idleProc (AEIdleUPP) - input
Reserved value, must be NULL.

AESend (OS/2) Parameter - filterProc

filterProc (AEFilterUPP) - input
Reserved value, must be NULL.

AESend (OS/2) Return Value - rc

rc (OSError) - returns
Return code.

noErr	No error.
eLenErr	The buffer was too big to send.
ERROR_NOT_ENOUGH_MEMORY	There was not enough memory.
connectionInvalid	A nonexistent signature or session ID was specified.
errAEEventNotHandled	The event was not handled by an OSA event handler.
errAEUnknownSendMode	An invalid sending mode was specified.
errAEWaitCanceled	The user canceled out of wait loop for reply or receipt.
errAETimeout	The OSA event timed out.
errAEUnknownAddressType	An unknown OSA event address type was specified.
ERROR_PROC_NOT_FOUND	There is no eligible process with specified process serial number.
errAESystemError	A general scripting system error occurred.
errAEMgrNotInitialized	

AESend (OS/2) - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to be sent.

reply ([OSAEvent *](#)) - output
The reply OSA event. This parameter is returned by the `AESend` method when `kAEMWaitReply` flag is specified in the `sendMode` parameter. If the `kAEQueueReply` flag is specified in the `sendMode` parameter and the event is not going to be directly dispatched, the reply OSA event is received in the event queue. If `kAENoReply` flag is specified, the reply OSA event returned by this function is a null-descriptor record. If `kAEMWaitReply` is specified in the `sendMode` parameter, the application is responsible for using the [AEDisposeDesc](#) method to dispose of the descriptor record returned in the `reply` parameter.

sendMode ([AESendMode](#)) - input
A flag indicating the appropriate mode.

These mode flags are divided into five categories. The `kAEAlwaysInteract`, `kAECanInteract`, and `kAENeverInteract` flags are used to communicate the user interaction preference to the server application, and only one of these can be set. The `kAECanSwitchLayers` flag is used to set the application switch mode. The flags `kAENoReply`, `kAEQueueReply`, and `kAEMWaitReply` specify the reply mode for an OSA event, and only one of these can be set. The last flag, `kAEWantReceipt`, sets the return receipt mode.

If your application is sending an event to itself, you can set the `kAEDontRecord` or the `kAEDontExecute` flag to prevent the event from being recorded or to ask the OSA Event Manager to record the event without your application actually receiving it.

The server is responsible for interrogating the following flags and honoring the client's requirements: `kAENeverInteract`, `kAECannotInteract`, `kAEAlwaysInteract`, and `kAECanSwitchLayers`.

kAEAlwaysInteract
The server application can interact with the user in response to the OSA event-by convention, whenever the server application normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user.

kAECanInteract
The server application can interact with the user in response to the OSA event-by convention, if the user needs to supply information to the server. This flag is the default when an OSA event is sent to a local application.

kAEDontExecute
Your application is sending an OSA event to itself for recording purposes only-that is, you want the OSA Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

kAEDontRecord
Your application is sending an event to itself but does not want the event recorded. When OSA event recording is on, the OSA Event Manager records a copy of every event your application sends to itself except for those events for which this flag is set.

kAENeverInteract
The server application should never interact with the user in response to the OSA event.

kAENoReply
Your application does not want a reply OSA event; the server processes your OSA event as soon as it has the opportunity. The OSA Event Manager returns immediately from the `AESend` function.

kAEQueueReply
Your application wants a reply OSA event; the reply appears in your event queue as soon as the server has the opportunity to process and respond to your OSA event. The OSA Event Manager returns immediately from the `AESend` function, except for direct dispatched events. Direct dispatched events cause the reply to be returned in the `reply` parameter of this function.

kAEMWaitReply
Your application wants a reply OSA event and is willing to give up the processor while waiting for the reply; for example, if the server application is on the same computer as your application, your application yields the processor to allow the server to respond to your OSA event. The OSA Event Manager creates a new thread for the calling process and blocks on this thread, waiting for the server to reply.

The process will continue to receive all messages (events) on its message queue.

kAEWantReceipt
The sender wants to receive a return receipt for this OSA event from the OSA Event Manager. (A return receipt means only that the receiving application accepted the OSA event; the OSA event may or may not be handled successfully after it is accepted.) If the receiving application does not send a returned receipt before the request times out, `AESend` returns `errAETimeout` as its function result.

sendPriority ([AESendPriority](#)) - input
A flag indicating the send priority. This parameter can be set to one of the following values:

kAENormalPriority OSA event is posted at the end of the event queue.
kAEHighPriority OSA event is posted at the front of the event queue.

timeOutInTicks (long) - input

The length of time, in ticks, that the client application is willing to wait for the reply or return receipt from the server application before timing out. This parameter can also be set to one of the constants in the following list:

kAEDefaultTimeout The OSA Event Manager provides an appropriate time-out duration.
kAENoTimeout The OSA event never times out.

idleProc (AEIdleUPP) - input

Reserved value, must be NULL.

filterProc (AEFilterUPP) - input

Reserved value, must be NULL.

rc (OSError) - returns

Return code.

noErr No error.
eLenErr The buffer was too big to send.
ERROR_NOT_ENOUGH_MEMORY There was not enough memory.
connectionInvalid A nonexistent signature or session ID was specified.
errAEEventNotHandled The event was not handled by an OSA event handler.
errAEUnknownSendMode An invalid sending mode was specified.
errAEWaitCanceled The user canceled out of wait loop for reply or receipt.
errAETimeout The OSA event timed out.
errAEUnknownAddressType An unknown OSA event address type was specified.
ERROR_PROC_NOT_FOUND There is no eligible process with specified process serial number.
errAESystemError A general scripting system error occurred.
errAEMgrNotInitialized

AESend (OS/2) - Remarks

If the OSA Event Manager cannot find a handler for a handler for an OSA event in either the application or system OSA event dispatch table, it returns the result code `errAEEventNotHandled` to the server applications (as the result of the [AEProcessOSAEvent](#) function). If the client application is waiting for a reply, the OSA Event Manager also returns this result code to the client.

The `AESend` method returns `noErr` if the OSA event was successfully sent by the OSA Event Manager. A `noErr` result from `AESend` does not indicate that the OSA event was successfully sent by the OSA Event Manager. If the handler returns a result code other than `noErr` and if the client is waiting for a reply, the result code is returned in the `keyErrorNumber` field of the *reply* parameter.

AESend detects when an application is sending an OSA event to itself. In this special case, a fast path is taken instead of placing the OSA event on the message queue. The applications event handler is called back directly during the send process. This avoids unnecessary overhead for factored applications which support recording. AESend will use WinPostMsg to perform the callback in this case.

The process of cloning an OSA event to support recording is also handled by AESend.

AESend (OS/2) - Example Code

This example shows how to send an event and wait for a reply for a time no longer than the default timeout. It is assumed that *theEvent* has already been built before calling AESend.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

OSErr myErr;                /* result code */
OSAEvent theEvent;          /* the OSA event to be sent */
OSAEvent theReply;          /* reply from the server */

myErr = AESend (&theEvent,
               &theReply,
               kAEWaitReply, /* mode */
               kAENormalPriority, /* priority */
               kAEDefaultTimeout, /* timeout */
               NULL, /* must be NULL */
               NULL); /* must be NULL */
```

AESend (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

AESetInteractionAllowed

AESetInteractionAllowed - Syntax

This function specifies your application's user interaction preferences for responding to an OSA event.

```
#define INCL_OSAEVENTS
#include <os2.h>
```

```

AEInteractAllowed    level;
OSErr                rc;    /* Return code. */

rc = AESetInteractionAllowed(level);

```

AESetInteractionAllowed Parameter - level

level ([AEInteractAllowed](#)) - input

The user interaction level to be set. This parameter can be set to one of the following values:

kAEInteractWithSelf

This flag allows the server application to interact with the user in response to an OSA event only when the client application and server application are the same-that is, only when the application is sending the OSA event to itself.

kAEInteractWithLocal

This flag allows the server application to interact with the user in response to an OSA event only if the client application is on the same computer as the server application; this is the default if the AESetInteractionAllowed function is not used.

kAEInteractWithAll

This flag allows the server application to interact with the user in response to an OSA event sent from any client application on any computer.

AESetInteractionAllowed Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

ERROR_INVALID_PARAMETER

The interaction level is invalid.

memError

AESetInteractionAllowed - Parameters

level ([AEInteractAllowed](#)) - input

The user interaction level to be set. This parameter can be set to one of the following values:

kAEInteractWithSelf

This flag allows the server application to interact with the user in response to an OSA event only when the client application and server application are the same-that is, only when the application is sending the OSA event to itself.

kAEInteractWithLocal

This flag allows the server application to interact with the user in response to an OSA event only if the client application is on the same computer as the server application; this is the default if the AESetInteractionAllowed function is not used.

kAEInteractWithAll

This flag allows the server application to interact with the user in response to an OSA event sent from any client application on any computer.

rc ([OSError](#)) - returns
Return code.

noErr No error.

ERROR_INVALID_PARAMETER
The interaction level is invalid.

memError

AESetInteractionAllowed - Remarks

This function sets the user interaction level for a server application's response to an OSA event.

AESetInteractionAllowed - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AESetTheCurrentEvent

AESetTheCurrentEvent - Syntax

This function specifies the OSA event to be handled.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent *theOSAEvent;
OSError rc; /* Return code. */

rc = AESetTheCurrentEvent(theOSAEvent);
```

AESetTheCurrentEvent Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to be handled.

AESetTheCurrentEvent Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
memError	

AESetTheCurrentEvent - Parameters

theOSAEvent ([const OSAEvent *](#)) - input
The OSA event to be handled.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
memError	

AESetTheCurrentEvent - Remarks

There is usually no reason for your application to use the AESetTheCurrentEvent function. Instead of calling this function, your application should let the OSA Event Manager set the current OSA event through the dispatch tables. If you need to avoid the dispatch tables, you must use the AESetTheCurrentEvent function only in the following way:

1. Your application suspends handling of an OSA event by calling the AESetTheCurrentEvent function.
2. Your application calls the AESetTheCurrentEvent function. This informs the OSA Event Manager that your application is handling the suspended OSA event. In this way, any routines that call the [AEGetTheCurrentEvent](#) function can ascertain which event is currently being handled.
3. When your application finishes handling the OSA event it calls the [AEResumeTheCurrentEvent](#) function with the value kAENoDispatch to tell the OSA Event Manager that the event has been processed and need not be dispatched.

AESetTheCurrentEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

AESizeOfAttribute

AESizeOfAttribute - Syntax

This function returns size and descriptor type of an OSA event attribute.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent    theOSAEvent;
AEKeyword         theAEKeyword; /* The keyword that specifies the attribute. */
DescType          *typeCode;    /* The descriptor type of the attribute. */
Size              *dataSize;    /* The length, in bytes, of the data in the attribute. */
OSErr             rc;           /* Return code. */

rc = AESizeOfAttribute(theOSAEvent, theAEKeyword,
                      typeCode, dataSize);
```

AESizeOfAttribute Parameter - theOSAEvent

theOSAEvent ([const OSAEvent](#)) - input
The OSA event containing the desired attribute.

AESizeOfAttribute Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the attribute.

AESizeOfAttribute Parameter - typeCode

typeCode ([DescType *](#)) - input
The descriptor type of the attribute.

AESizeOfAttribute Parameter - dataSize

dataSize ([Size *](#)) - input

The length, in bytes, of the data in the attribute.

AESizeOfAttribute Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfAttribute - Parameters

theOSAEvent ([const OSAEvent](#)) - input
The OSA event containing the desired attribute.

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the attribute.

typeCode ([DescType *](#)) - input
The descriptor type of the attribute.

dataSize ([Size *](#)) - input
The length, in bytes, of the data in the attribute.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfAttribute - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AESizeOfDescData (OS/2)

AESizeOfDescData (OS/2) - Syntax

This function returns the size of the descriptor data that is contained in the specified descriptor record.

```
#define INCL_OSAEVENTS
#include <os2.h>

const AEDesc      *theAEDesc;
Size              *dataSize;
OSErr             rc;          /* Return code. */

rc = AESizeOfDescData(theAEDesc, dataSize);
```

AESizeOfDescData (OS/2) Parameter - theAEDesc

theAEDesc (`const AEDesc *`) - input
A pointer to the descriptor record whose size is to be returned.

AESizeOfDescData (OS/2) Parameter - dataSize

dataSize (`Size *`) - in/out
A pointer to the size, in bytes, of the data referenced by the specified descriptor record.

AESizeOfDescData (OS/2) Return Value - rc

rc (`OSErr`) - returns
Return code.

noErr
No error.

AESizeOfDescData (OS/2) - Parameters

theAEDesc (`const AEDesc *`) - input
A pointer to the descriptor record whose size is to be returned.

dataSize (`Size *`) - in/out

A pointer to the size, in bytes, of the data referenced by the specified descriptor record.

rc ([OSError](#)) - returns
Return code.

noErr
No error.

AESizeOfDescData (OS/2) - Remarks

This method can be used to determine the size of the buffer needed when calling the [AEGetDescData](#) method.

AESizeOfDescData (OS/2) - Example Code

This example uses the `AESizeOfDescData` method to determine the size of the data in a descriptor record. This is useful when extracting unterminated strings from descriptor records of type `typeChar`.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

OSError    myErr;                /* result code */
AEDesc     theDesc;              /* descriptor record */
Size       dataSize, actualSize; /* size of data */
DescType   typeCode;             /* descriptor type */
Ptr        theData;              /* buffer pointer */

myErr = AESizeOfDescData(&theDesc,
                        &dataSize);

if(myErr == noErr)
{
    theData = (Ptr) malloc(dataSize);

    myErr = AEGetDescData(&theDesc,
                          &typeCode,
                          theData,
                          dataSize,
                          &actualSize);
}
```

AESizeOfDescData (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

AESizeOfKeyDesc

AESizeOfKeyDesc - Syntax

This function returns the size and descriptor type of a keyword-specified descriptor record in an AE record.

```
#define INCL_OSAEVENTS
#include <os2.h>

AERecord      *theAERecord;
AEKeyword     theAEKeyword;
DescType      *typeCode;
Size          *dataSize;
OSErr         rc;          /* Return code. */

rc = AESizeOfKeyDesc(theAERecord, theAEKeyword,
                    typeCode, dataSize);
```

AESizeOfKeyDesc Parameter - theAERecord

theAERecord ([AERecord *](#)) - input
The AE record containing the desired keyword-specified descriptor record.

AESizeOfKeyDesc Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired descriptor record.

AESizeOfKeyDesc Parameter - typeCode

typeCode ([DescType *](#)) - output
The descriptor type of the keyword-specified descriptor record.

AESizeOfKeyDesc Parameter - dataSize

dataSize ([Size *](#)) - output
The length, in bytes of the data in the keyword-specified descriptor record.

AESizeOfKeyDesc Return Value - rc

rc ([OSError](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfKeyDesc - Parameters

theAERecord ([AERecord *](#)) - input
The AE record containing the desired keyword-specified descriptor record.

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired descriptor record.

typeCode ([DescType *](#)) - output
The descriptor type of the keyword-specified descriptor record.

dataSize ([Size *](#)) - output
The length, in bytes of the data in the keyword-specified descriptor record.

rc ([OSError](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfKeyDesc - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AESizeOfNthItem

AESizeOfNthItem - Syntax

This function returns the size and descriptor type of a descriptor record in a descriptor list.

```
#define INCL_OSAEVENTS
#include <os2.h>

const AEDescList    *theAEDescList;
long                index;
DescType            *typeCode;
Size                *dataSize;
OSErr               rc;          /* Return code. */

rc = AESizeOfNthItem(theAEDescList, index,
                    typeCode, dataSize);
```

AESizeOfNthItem Parameter - theAEDescList

theAEDescList (`const AEDescList *`) - input
The descriptor list containing the descriptor record.

AESizeOfNthItem Parameter - index

index (`long`) - input
The position of the descriptor record in the list (for example, 2 specifies the second descriptor record).

AESizeOfNthItem Parameter - typeCode

typeCode (`DescType *`) - output
The descriptor type of the descriptor record.

AESizeOfNthItem Parameter - dataSize

dataSize ([Size *](#)) - output
The length, in bytes, of the data in the descriptor record.

AESizeOfNthItem Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfNthItem - Parameters

theAEDescList ([const AEDescList *](#)) - input
The descriptor list containing the descriptor record.

index (long) - input
The position of the descriptor record in the list (for example, 2 specifies the second descriptor record).

typeCode ([DescType *](#)) - output
The descriptor type of the descriptor record.

dataSize ([Size *](#)) - output
The length, in bytes, of the data in the descriptor record.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfNthItem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AESizeOfParam

AESizeOfParam - Syntax

This function returns the size and descriptor type of an OSA event parameter.

```
#define INCL_OSAEVENTS
#include <os2.h>

OSAEvent      *theOSAEvent;
AEKeyword      theAEKeyword;
DescType      *typeCode;
Size          *dataSize;
OSErr          rc;          /* Return code. */

rc = AESizeOfParam(theOSAEvent, theAEKeyword,
                  typeCode, dataSize);
```

AESizeOfParam Parameter - theOSAEvent

theOSAEvent ([OSAEvent](#) *) - input
The OSA event containing the parameter.

AESizeOfParam Parameter - theAEKeyword

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

AESizeOfParam Parameter - typeCode

typeCode ([DescType](#) *) - output
The descriptor type of the OSA event parameter.

AESizeOfParam Parameter - dataSize

dataSize ([Size](#) *) - output
The length, in bytes, of the data in the OSA event parameter.

AESizeOfParam Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfParam - Parameters

theOSAEvent ([OSAEvent *](#)) - input
The OSA event containing the parameter.

theAEKeyword ([AEKeyword](#)) - input
The keyword that specifies the desired parameter.

typeCode ([DescType *](#)) - output
The descriptor type of the OSA event parameter.

dataSize ([Size *](#)) - output
The length, in bytes, of the data in the OSA event parameter.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errAEDescNotFound	The descriptor record was not found.
errAENotAEDesc	The descriptor record was invalid.
errAEReplyNotArrived	The reply has not yet arrived.

AESizeOfParam - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

AE_suspendTheCurrentEvent

AE_suspendTheCurrentEvent - Syntax

This function suspends the processing of the OSA event that is currently being handled.

```
#define INCL_OSAEVENTS
#include <os2.h>

const OSAEvent    *theOSAEvent;
OSErr            rc;          /* Return code. */

rc = AESuspendTheCurrentEvent(theOSAEvent);
```

AESuspendTheCurrentEvent Parameter - theOSAEvent

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event whose handling is to be suspended. Although the OSA Event Manager does not need this parameter to identify the OSA event currently being handled, providing it is a safeguard that you are suspending the correct OSA event.

AESuspendTheCurrentEvent Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

AESuspendTheCurrentEvent - Parameters

theOSAEvent ([const OSAEvent *](#)) - input

The OSA event whose handling is to be suspended. Although the OSA Event Manager does not need this parameter to identify the OSA event currently being handled, providing it is a safeguard that you are suspending the correct OSA event.

rc ([OSErr](#)) - returns
Return code.

noErr
No error.

AESuspendTheCurrentEvent - Remarks

After a server application makes a successful call to the AESuspendTheCurrentEvent function, it is no longer required to return a result or a reply for the OSA event that was being handled. It can, however, return a result if it later calls the [AEResumeTheCurrentEvent](#) function to resume event processing. The OSA Event Manager does not automatically dispose of OSA events that have been suspended or their default replies. (The OSA Event Manager does, however, automatically dispose of a previously suspended OSA event and its default reply if the server later resumes processing of the OSA event by calling the [AEResumeTheCurrentEvent](#) function.) If your server application does not resume processing of a suspended OSA event, it is responsible for using the [AEDisposeDesc](#) function to dispose of both the OSA event and its default reply when your application has finished using them.

Special Considerations

If your application suspends handling of an OSA event it sends to itself, the OSA Event Manager immediately returns from the [AESend](#) call with the error code `errAETimeout`, regardless of whether the `kAEQueueReply`, `kAEWaitReply`, or `kAENoReply` flags were set, even if the `timeOutInTicks` parameter is set to `kNoTimeOut`. The routine calling [AESend](#) should take the timeout error as confirmation that the event was sent. As with other calls to [AESend](#) that return a timeout error, the handler continues to process the event nevertheless. The handler's reply, if any, is provided in the reply event when the handling is completed. The OSA Event Manager provides no notification that the reply is ready. If no data has yet been placed in the reply event, the OSA Event Manager returns `errAEReplyNotArrived` when your application attempts to extract data from the reply.

AESuspendTheCurrentEvent - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

AETerminate (OS/2)

AETerminate (OS/2) - Syntax

This function unregisters applications from the OSA Event Manager when the application ends.

```
#define INCL_OSAEVENTS
#include <os2.h>

OSErr rc; /* Return code. */

rc = AETerminate();
```

AETerminate (OS/2) Return Value - rc

rc ([OSErr](#)) - returns
Return code.

`noErr`

No error.

AETerminate (OS/2) - Parameters

rc (**OSErr**) - returns
Return code.

noErr

No error.

AETerminate (OS/2) - Remarks

The OSA Event Manager must clean up internal structures when an application ends. This function calls an exist list routine to perform this clean-up during process termination.

The use of this function is optional because clean-up is performed automatically by the exit list routine during process termination.

AETerminate (OS/2) - Example Code

This example shows how to deregister an application with the OSA Event Manager during the application termination procedures.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#define INCL_WIN
#include <os2.h>

int main(int argc, char *argv[])
{
    HAB hab;                                /* anchor block handle */
    HMQ hmq;                                /* message queue handle */
    HWND hwndFrame;                          /* frame window handle */
    QMSG qmsg;                              /* message from queue */

    ...

    while(WinGetMsg(hab, &qmsg, (HWND)0, 0, 0))
    {
        WinDispatch(hab, &qmsg);
    }

    AETerminate();
    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
    return 0;
}
```

AETerminate (OS/2) - Topics

Select an item:

OSAInstallApplication (OS/2) (C)

OSAInstallApplication (OS/2) (C) - Syntax

This function add an application to the OSA registration data base.

```
#define INCL_OSAEVENTS
#define INCL_OSAAPI
#include <os2.h>

PSZ      description;
PSZ      name;
PSZ      AETEFFileName;
USHORT   usSCSZ;
ULONG    AppType;
ULONG    Reserved1;
ULONG    Reserved2;
OSAError rc;          /* Return code. */

rc = OSAInstallApplication(description, name,
                           AETEFFileName, usSCSZ, AppType, Reserved1,
                           Reserved2);
```

OSAInstallApplication (OS/2) (C) Parameter - description

description ([PSZ](#)) - input

The descriptive name of the application; for example, "Lotus 123".

OSAInstallApplication (OS/2) (C) Parameter - name

name ([PSZ](#)) - input

The name of the application' executable file; for example, "x:\lotus\123.exe".

OSAInstallApplication (OS/2) (C) Parameter - AETEFFileName

AETEFileName (**PSZ**) - input
The name and path of the file containing the AETE resource.

OSAInstallApplication (OS/2) (C) Parameter - usSCSZ

usSCSZ (**USHORT**) - input
The **scsz** flag settings for this application. The flag can be set to any of the following values:

kLaunchToGetTerminology
This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the application's **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kAlwaysSendSubject
When this bit is set, scripting components and other applications that sent events to this application must include a subject attribute in the event.

OSAInstallApplication (OS/2) (C) Parameter - AppType

AppType (**ULONG**) - input
The application type.

OSA_APPLICATION
A PM application
OSA_PARTHANDLER
An OpenDoc part handler

OSAInstallApplication (OS/2) (C) Parameter - Reserved1

Reserved1 (**ULONG**) - input
Reserved value, must be 0.

OSAInstallApplication (OS/2) (C) Parameter - Reserved2

Reserved2 (**ULONG**) - input
Reserved value, must be 0.

OSAInstallApplication (OS/2) (C) Return Value - rc

rc ([OSAEError](#)) - returns
Return code.

errAEAppAlreadyInstalled
The application or part handler is already in the registration data base.

errAEBadParm
One or more of the parameters passed in are invalid, or the reserved fields are not set to zero.

OSAInstallApplication (OS/2) (C) - Parameters

description ([PSZ](#)) - input
The descriptive name of the application; for example, "Lotus 123".

name ([PSZ](#)) - input
The name of the application's executable file; for example, "x:\lotus\123.exe".

AETEFileName ([PSZ](#)) - input
The name and path of the file containing the AETE resource.

usSCSZ ([USHORT](#)) - input
The **scsz** flag settings for this application. The flag can be set to any of the following values:

kLaunchToGetTerminology
This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the application's **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kAlwaysSendSubject
When this bit is set, scripting components and other applications that sent events to this application must include a subject attribute in the event.

AppType ([ULONG](#)) - input
The application type.

OSA_APPLICATION
A PM application
OSA_PARTHANDLER
An OpenDoc part handler

Reserved1 ([ULONG](#)) - input
Reserved value, must be 0.

Reserved2 ([ULONG](#)) - input
Reserved value, must be 0.

rc ([OSAEError](#)) - returns
Return code.

errAEAppAlreadyInstalled
The application or part handler is already in the registration data base.

errAEBadParm
One or more of the parameters passed in are invalid, or the reserved fields are not set to zero.

OSAInstallApplication (OS/2) (C) - Remarks

Applications and part handlers use this API at install time to register it as being OSA aware and install corresponding AETE and SCSZ resources. The reserved parameters must be set to zero.

Note: The name of the application or part handler .EXE or .DLL files should include the path.

The `kAlwaysSendSubject` bit in the `usSCSZ` parameter is defined to give applications more control over which handlers are used when an event is processed. If server applications want the target object to always handle an event, regardless of what the direct object of the event contains, these applications must set the `kAlwaysSendSubject` bit. Even if the direct parameter of the event is an object specifier, all events sent to these applications must include a subject attribute which identifies the target of the event.

OSAInstallApplication (OS/2) (C) - Example Code

This example adds an application to the OSA registration data base.

```
OSAError rc;

/* Register my application in the OSA registration data base. */
rc = OSAInstallApplication( "My OSA App",    /* descriptive app name */
                           "myapp.exe",     /* app's exe name */
                           "myapp.aet",     /* app's aete file name */
                           0,               /* app's SCSZ (0 is default) */
                           OSA_APPLICATION, /* app type */
                           0, 0)           /* reserved, must be zero */
```

OSAInstallApplication (OS/2) (C) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

OSAInstallApplication

OSAInstallApplication - Example Code

The following Object REXX example installs an OSA application and a part handler.

```
/* ***** */
/* APPLICATION_TYPE -- #define OSA_APPLICATION 0 */
/* #define OSA_PARTHANDLER 1 */
/* ***** */

AETE = 'myapp.aet'
AETE2 = 'mypart.aet'
APP = 0
```

```

PART = 1
RESERVED = 0

call RxFuncAdd 'OSALoadFuncs', 'OPENDOC', 'OSALoadFuncs'

call OSALoadFuncs

/* Install "My OSA App". */
rc = OSAInstallApplication('My OSA App', 'myapp.exe', AETE, 0, APP, RESERVE
SAY " OSAInstallApp rc = " rc

/* Install "My OSA Part". */
rc = OSAInstallApplication('My OSA Part', 'mypart.dll', AETE2, 0, PART, RES
SAY " OSAInstallApp rc = " rc

call OSAUnloadFuncs

RETURN

```

OSAInstallApplication - Purpose

This function is called by applications and part handlers at installation time to register the application as being OSA-aware and install its corresponding AETE and SCSZ resources.

OSAInstallApplication Keyword - description

description

The descriptive name of the application; for example, "Lotus 123".

OSAInstallApplication Keyword - name

name

The name of the application's executable file; for example, "x:\lotus\123.exe".

OSAInstallApplication Keyword - AETEFileName

AETEFileName

The name and path of the file containing the AETE resource.

OSAInstallApplication Keyword - usSCSZ

usSCSZ

The **scsz** flag settings for this application. The flag can be set to any of the following values:

OSAInstallApplication Keyword - AppType

AppType

The application type.

OSAInstallApplication Keyword - Reserved1

Reserved1

Reserved value, must be 0.

OSAInstallApplication Keyword - Reserved2

Reserved2

Reserved value, must be 0.

OSAInstallApplication - Keywords

description

The descriptive name of the application; for example, "Lotus 123".

name

The name of the application' executable file; for example, "x:\lotus\123.exe".

AETEFileName

The name and path of the file containing the AETE resource.

usSCSZ

The **scsz** flag settings for this application. The flag can be set to any of the following values:

AppType

The application type.

Reserved1

Reserved value, must be 0.

Reserved2

Reserved value, must be 0.

OSAInstallApplication - Syntax

```
OSAInstallApplication ( description , name , AETEFileName ,  
                        usSCSZ , AppType ,  
                        Reserved1 , Reserved2 )
```

Examples

OSAInstallApplication - Remarks

This function returns one of the following return codes:

errAEAppAlreadyInstalled The application or part handler is already in the registration data base.

errAEBadParm One or more of the parameters passed in are invalid, or the reserved fields were not set to zero.

Applications and part handlers use this API at installation time to register it as being OSA-aware and install corresponding AETE and SCSZ resources. The reserved parameters must be set to zero.

Note: The name of the application or part handler .EXE or .DLL files should be included the path.

OSAInstallApplication - Topics

Select an item:

[Purpose](#)

[Syntax](#)

[Keywords](#)

[Remarks](#)

[Example Code](#)

[Glossary](#)

OSAListApplications (OS/2)

OSAListApplications (OS/2) - Syntax

This function is called by application to get a list of applications and part handlers that are OSA aware.

```
#define INCL_OSAEVENTS  
#define INCL_OSAAPI  
#include <os2.h>
```

```

PSZ      AppNames;
PULONG   psize;
ULONG    apptype;
OSAError rc;      /* Return code. */

rc = OSAListApplications(AppNames, psize,
    apptype);

```

OSAListApplications (OS/2) Parameter - AppNames

AppNames (**PSZ**) - input

A buffer containing a continuous list of null terminated names. This parameter can be set to NULL to get the required size of the buffer.

OSAListApplications (OS/2) Parameter - psize

psize (**PULONG**) - in/out

On input, this parameter is the size of the *AppNames* buffer. On output, the size of the data written to *AppNames* is returned.

OSAListApplications (OS/2) Parameter - apptype

apptype (**ULONG**) - in/out

The application type.

OSA_ALL	Both PM applications and OpenDoc part handlers
OSA_APPLICATION	A PM application
OSA_PARTHANDLER	An OpenDoc part handler

OSAListApplications (OS/2) Return Value - rc

rc (**OSAError**) - returns

Return code.

errAEBufferTooSmall	The size of the buffer is not large enough to hold the data to be returned.
errAEBadParm	One or more of the parameters passed in are invalid, or the reserved fields are not set to zero.

OSAListApplications (OS/2) - Parameters

AppNames (PSZ) - input

A buffer containing a continuous list of null terminated names. This parameter can be set to NULL to get the required size of the buffer.

psize (PULONG) - in/out

On input, this parameter is the size of the *AppNames* buffer. On output, the size of the data written to *AppNames* is returned.

apptype (ULONG) - in/out

The application type.

OSA_ALL

Both PM applications and OpenDoc part handlers

OSA_APPLICATION

A PM application

OSA_PARTHANDLER

An OpenDoc part handler

rc (OSAEError) - returns

Return code.

errAEBufferTooSmall

The size of the buffer is not large enough to hold the data to be returned.

errAEBadParm

One or more of the parameters passed in are invalid, or the reserved fields are not set to zero.

OSAListApplications (OS/2) - Example Code

This example prints a list of registered OSA-aware applications.

```
PSZ          pszBuffer = NULL;
```

```
PSZ          pszTemp = NULL;
```

```
ULONG        size = 0;
```

```
ULONG        appType;
```

```
OSAEError     rc;
```

```
/* Set appType to look for applications */
/* (could also look for parts, OSA_PARTHANDLER, or both, OSA_ALL). */
appType = OSA_APPLICATION;
```

```
/* First call list applications to get the required buffer size. */
rc = OSAListApplications(NULL, &size, appType);
```

```
if(size)
{
    /* Allocate required buffer. */
    pszBuffer = (char *) malloc(size);

    /* Now, call list applications with our buffer. */
    rc = OSAListApplications(pszBuffer, &size, appType);
}
```

```
pszTemp = pszBuffer; /* Save ptr for free. */
```

```
/* Print out the name of each application returned. */
while(strlen(pszBuffer))
{
    printf("App Name = %s\n", pszBuffer);
    pszBuffer = pszBuffer + (strlen(pszBuffer) + 1);
}
```

```
}  
  
free (pszTemp) ;
```

OSAListApplications (OS/2) - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

OSARemoveApplication (OS/2) (C)

OSARemoveApplication (OS/2) (C) - Syntax

This function is called by applications and part handlers at un-install time to remove itself from the list of OSA aware applications, and uninstall AETE and SCSZ resources.

```
#define INCL_OSAEVENTS  
#define INCL_OSAAPI  
#include <os2.h>  
  
PSZ      description;  
OSAError rc;          /* Return code. */  
  
rc = OSARemoveApplication(description);
```

OSARemoveApplication (OS/2) (C) Parameter - description

description ([PSZ](#)) - input

The descriptive name of the application to be removed; for example, "Lotus 123".

OSARemoveApplication (OS/2) (C) Return Value - rc

rc ([OSAError](#)) - returns
Return code.

errAEAppNotFound

The specified application or part handler is not found in the registration data base.

errAEBadParm

One or more of the parameters passed are invalid.

OSARemoveApplication (OS/2) (C) - Parameters

description ([PSZ](#)) - input

The descriptive name of the application to be removed; for example, "Lotus 123".

rc ([OSAError](#)) - returns

Return code.

errAEAppNotFound

The specified application or part handler is not found in the registration data base.

errAEBadParm

One or more of the parameters passed are invalid.

OSARemoveApplication (OS/2) (C) - Example Code

This example removes an application and its **aete** and **scsz** resources from the OSA registration data base.

```
OSAError      rc;
```

```
/* Remove my application from the OSA registration data base; */  
/* this removes my app's AETE and SCSZ information as well. */  
rc = OSARemoveApplication("My OSA App");
```

OSARemoveApplication (OS/2) (C) - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

OSARemoveApplication

OSARemoveApplication - Example Code

This Object REXX example uninstalls an application.

```
call RxFuncAdd 'OSALoadFuncs', 'OPENDOC', 'OSALoadFuncs'

call OSALoadFuncs

/* Uninstall "My OSA App". */
rc = OSARemoveApplication('My OSA App')
SAY " OSARemoveApp rc = " rc

call OSAUnloadFuncs

RETURN
```

OSARemoveApplication - Purpose

This REXX function is called by applications and part handlers at uninstall time to remove itself from the list of OSA-aware applications, and uninstall AETE and SCSZ resources.

OSARemoveApplication Keyword - description

description

The descriptive name of the application to be removed; for example, "Lotus 123".

OSARemoveApplication - Keywords

description

The descriptive name of the application to be removed; for example, "Lotus 123".

OSARemoveApplication - Syntax

```
OSARemoveApplication ( description )
```

Examples

OSARemoveApplication - Remarks

This function returns one of the following return codes:

errAEAppNotFound

The specified application or part handler is not found in the registration data base.

errAEBadParm

One or more of the parameters passed are invalid.

OSARemoveApplication - Topics

Select an item:

[Purpose](#)

[Syntax](#)

[Keywords](#)

[Remarks](#)

[Example Code](#)

[Glossary](#)

OSA Classes and Methods

This chapter covers the classes which provide the OSA interfaces for applications to support embedded scripts. This allows applications to support "smart objects," such as menu items or buttons which have scripts attached to them, to provide customized functions.

IMPORTANT: Methods have two additional parameters that are not shown. The first parameter is a pointer to the object on which the method is being invoked. This does not need to be specified if you are using C++. The second parameter is a pointer to the environment variable. For example, the syntax of the [ComponentManager::CloseComponent](#) method in the SOM language is as follows:

```
OSErr CloseComponent (Component *somSelf, Environment *ev, Component
```

If you are using C++, the syntax is as follows:

```
OSErr CloseComponent (Environment *ev, Component *theComponentInstar
```

The Exception Handling is not shown unless exceptions exists for that method. The Override Policy is also not shown unless it is different than the default override policy in which a derived class can override the method and can call the parent's copy of the method from within its own copy.

Application Types

Applications and OpenDoc parts can implement different levels of scripting support. They range from the scriptable applications which are relatively simple to implement to the more complex support required for recordable applications.

Scriptable Applications

Scriptable applications conform to one or more of the OSA Event Registry suites. In addition, they have an **aete** resource that states their conformance to or deviations from the registry. They can be driven by means of their semantic interface from other applications, OpenDoc parts, or scripting languages.

Attachable Applications

An application or OpenDoc part that is attachable allows end users to attach or embed scripts that are associated with objects, such as menu items or buttons. This allows the application to be "tinkerable" or customized to meet unique needs.

Recordable Applications

Recordable applications have a factored structure. The user interface code is separated from the code which implements the application's core functions. Semantic actions which occur from the user interface are converted into OSA events and sent back to the application for processing by means of the core engine.

When another application, such as a scripting language editor, requests the OSA Event Manager to record, it receives copies of these OSA event that the recorded application is sending to itself. They can be converted into source code and recorded in a script for later execution.

Users who have little or no knowledge of a particular scripting language can record simple scripts. More proficient users can edit and combine recorded scripts to meet their needs.

In addition, recording is a mechanism for implementing intelligent agents. *Intelligent agents* can observe the way people use the system and can adopt the system's behavior to anticipate the user's needs. This feature is very useful in environments with a "pro-active" user interface.

SOM

The OSA scripting language support is implemented as SOM classes. This allows the code to be ported to other platforms, such as AIX, with minimal amount of effort.

The scripting component APIs are implemented as methods of an abstract SOM class which scripting components can subclass.

Scripting Component Registration

Scripting components are identified by a *component type code* and a *subtype code*. The component type code for all scripting components is:

```
kOSAComponentType = 'osa '
```

A *manufacturer code* for IBM is registered with CI Labs. This maintains cross-platform consistency.

A subtype code for each scripting component is assigned by CI Labs. Scripting component developers should register their OS/2 components with CI Labs Registry Group to ensure that the subtype codes are compatible across platforms. A subtype code of **OREX** is registered for the OS/2 Object REXX scripting component.

Scripting components are added to the component registration data base using the [InstallComponent](#) method of the [ComponentManager](#) class. C and Object REXX versions of this method are available (see [Component Manager Functions](#)).

Classes Hierarchy

This chapter contains an alphabetic listing of the OSA object classes and their methods. These sections contain technical reference information. For information on the System Object Model (SOM), see the *System Object Model Guide and Reference*.

The following list describes some terminology used in the following sections:

class	A way of categorizing objects based on their behavior and shape. A class is, in effect, a definition of a generic object. In SOM, a class is a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior (more precisely, all of the objects manufactured by a class have the same memory layout and share a common set of methods). New classes can be defined in terms of existing classes through a technique known as <i>inheritance</i> .
class method	A class method of class <X> is a method provided by the metaclass of class <X>. Class methods are executed without requiring any instances of class <X> to exist and are frequently used to create instances.
inheritance	The technique of specifying the shape and behavior of one class (called a <i>subclass</i>) as incremental differences from another class (called the <i>parent class</i> or <i>superclass</i>). The subclass inherits the superclass' state representation and methods and can provide additional data elements and methods. The subclass also can provide new functions with the same method names used by the superclass. Such a subclass method is said to override the superclass method and will be selected automatically by method resolution on subclass instances. An overriding method can elect to call upon the superclass' method as part of its own implementation.
instance	(Or object instance). A specific object, as distinguished from the abstract definition of an object referred to as its class.
instance method	A method that is valid for a particular object.
metaclass	A class whose instances are all classes. In SOM, any class descended from SOMClass is a metaclass. The methods of a metaclass are sometimes called "class" methods.
method	One of the units that make up the behavior of an object. A method is a combination of a function and a name, such that many different functions can have the same name. Which function the name refers to, at any point, depends on the object that is to execute the method and is the subject of method resolution.
object	<p>The elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way of associating specific data values with a specific set of named functions (called <i>methods</i>) for a period of time (referred to as the <i>lifetime</i> of the object). The data values of an object are referred to as its <i>state</i> . In SOM, objects are created by other objects called <i>classes</i> . The specification of what comprises the set of functions and data elements that make up an object is referred to as the <i>definition</i> of a class.</p> <p>SOM objects offer a high degree of <i>encapsulation</i> . This property permits many aspects of the implementation of an object to change without affecting client programs that depend on the object's behavior.</p>
object class	See <i>class</i> .
object instance	See <i>instance</i> .
subclass	A class that inherits from another class. See <i>inheritance</i> .
superclass	A class from which another class inherits. See <i>inheritance</i> .

OSA Class Hierarchy

The following figure lists the predefined OSA object classes in a hierarchical order. Each branch in the tree represents an immediate descendant (subclass) of an OSA object class. The predefined SOM object class, SOMObject, is the root class for all SOM object classes, including all OSA object classes.

CLASS NAME	CLASS DEFINITION FILE
SOMObject	somobj.idl
SOMClassMgr	somcm.idl
SOMClass	somcls.idl
Component	comp.idl
OSAScriptingComponent	osasc.idl
GenericScriptingComponent	osagsc.idl

Instances of some OSA object classes cannot be created as an OSA object. These classes are provided as base classes that provide support for descendant classes that can have instances created. Other classes are SOM classes and are described in more detail in the *System Object Model Guide and Reference*. These classes include:

SOMObject	This is the SOM root class. All SOM classes must be descended from SOMObject. An OSA object of this class cannot be created.
SOMClass	This is the SOM metaclass, that is, the instances of this class are class objects. An OSA object of this class cannot be created.
SOMClassMgr	This is the SOM class manager class. An OSA object of this class cannot be created.

Component

Class Definition File: COMP.IDL

Class Hierarchy

SOMObject
 Component

Description

The component class is a generic base class for all components, not just scripting components. It can also be used for OpenDoc translation components.

Methods

The following list shows the methods defined by the Component class:

- [GetComponentInstanceError](#)
- [GetComponentVersion](#)
- [SetComponentInstanceError](#)

Overridden Methods

There are currently no methods overridden by the Component class.

GetComponentInstanceError

GetComponentInstanceError - Syntax

This method returns the last error generated by a specific connection to a component.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

OSErr    rc;

rc = GetComponentInstanceError();
```

GetComponentInstanceError Return Value - rc

rc ([OSErr](#)) - returns
The last error set by this component.

GetComponentInstanceError - Parameters

rc ([OSErr](#)) - returns
The last error set by this component.

GetComponentInstanceError - Remarks

Once an error code is retrieved, the Component Manager clears the error code from the connection. If you want to retain that error value, you must save it in your application's local storage.

GetComponentInstanceError - Example Code

This example returns the last error generated by a specific connection to a component.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
OSErr            err = 0;

/* create an instance of the component manager */
cmgr = new ComponentManager;

/* create an instance of Object Rexx scripting component */
sc = (OSAScriptingComponent *) cmgr->OpenDefaultComponent(ev, kOSAComponent
                                                         kObjectRexxSubtyp

/* get the last error generated by the component */
err = sc->GetComponentInstanceError(ev)
```

GetComponentInstanceError - Topics

Class:
Component

Select an item:

GetComponentVersion

GetComponentVersion - Syntax

This method returns a component's version number.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ODULong    rc;

rc = GetComponentVersion();
```

GetComponentVersion Return Value - rc

rc ([ODULong](#)) - returns
The version number of the component.

GetComponentVersion - Parameters

rc ([ODULong](#)) - returns
The version number of the component.

GetComponentVersion - Remarks

This function returns a long integer containing the version number of the specified component. The high-order 16 bits represent the major version, and the low-order 16-bits represent the minor version. The major version specifies the component specification level; the minor version specifies a particular implementation's version number.

GetComponentVersion - Example Code

This example prints a component's version number.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;

/* create an instance of the component manager */
cmgr = new ComponentManager;

/* create an instance of Object Rexx scripting component */
sc = (OSAScriptingComponent *) cmgr->OpenDefaultComponent(ev, kOSAComponent
                                                         kObjectRexxSubtyp

/* print out the version number of this component */
printf("Component Version = %d\n",sc->GetComponentVersion(ev));
```

GetComponentVersion - Topics

Class:

Component

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

SetComponentInstanceError

SetComponentInstanceError - Syntax

This method sets the error associated with a component to a specified value.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

OSErr theError;

SetComponentInstanceError(theError);
```

SetComponentInstanceError Parameter - theError

theError ([OSErr](#)) - input

The new value for the current error. The Component Manager uses this value to set the current error for the connection.

SetComponentInstanceError - Return Value

None.

SetComponentInstanceError - Parameters

theError ([OSErr](#)) - input

The new value for the current error. The Component Manager uses this value to set the current error for the connection.

None.

SetComponentInstanceError - Remarks

The Component Manager maintains error state information for all currently active components. Although a component usually returns error information as its result, some requests require that the component return other information as its result. In this case, a component can choose to use the `SetComponentInstanceError` method to report its latest error state to the Component Manager. The Component Manager uses this error information to set the current error value for the appropriate connection. Applications can then retrieve this error information by calling the [GetComponentInstanceError](#) method. The documentation for the component should specify how the component indicates errors.

SetComponentInstanceError - Example Code

This example sets the error associated with a component to 1.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
OSErr            err = 0;

/* create an instance of the component manager */
cmgr = new ComponentManager;

/* create an instance of Object Rexx scripting component */
sc = (OSAScriptingComponent *) cmgr->OpenDefaultComponent(ev, kOSAComponent
                                                         kObjectRexxSubtyp

/* set the error associated with a component */
err = 1;
sc->SetComponentInstanceError(ev, err)
```

SetComponentInstanceError - Topics

Class:

Component

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

ComponentManager

Class Definition File: COMPMGR.IDL

Class Hierarchy

SOMObject

ComponentManager

Description

The ComponentManager class implements the support for obtaining a component's registration information from the registration data base, searching the data base for specific components and instantiating a component object.

The ComponentManager deals with "pieces of code" called *components*, which provide a defined set of services to clients. Components can register their type, subtype, level of service (capabilities), and manufacturer during system initialization. All components with a given type and subtype must support a standard set of interfaces. The ComponentManager implements support for obtaining a component's registration information from the registration data base, searching the data base for specific components and instantiating component objects.

Methods

The following list shows the methods defined by the ComponentManager class:

- [CloseComponent](#)
- [CountComponents](#)
- [FindNextComponent](#)
- [GetComponentInfo](#)
- [GetComponentRefcon](#)
- [InstallComponent](#)
- [OpenComponent](#)
- [OpenDefaultComponent](#)
- [SetComponentRefcon](#)
- [UninstallComponent](#)

Overridden Methods

There are currently no methods overridden by the ComponentManager class.

CloseComponent

CloseComponent - Syntax

This method closes a component.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

Component      *theComponentInstance;
OSErr          rc;          /* Return code. */

rc = CloseComponent(theComponentInstance);
```

CloseComponent Parameter - theComponentInstance

theComponentInstance (Component *) - input

The component instance to be closed. The component instance can be obtained from the [OpenComponent](#) or [OpenDefaultComponent](#) methods.

CloseComponent Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errCMInvalidComponentInstance

The component instance is invalid.

CloseComponent - Parameters

theComponentInstance (Component *) - input

The component instance to be closed. The component instance can be obtained from the [OpenComponent](#) or [OpenDefaultComponent](#) methods.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errCMInvalidComponentInstance

The component instance is invalid.

CloseComponent - Remarks

This method closes only a single connection. If your application has several connections to a single component, you must call the CloseComponent method once for each connection.

CloseComponent - Example Code

This example closes an Object REXX scripting component using the CloseComponent function.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
OSERR             err = 0;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Create an instance of Object REXX scripting component. */
sc = (OSAScriptingComponent *) cmgr->OpenDefaultComponent(ev, kOSAComponent
                                                         kObjectRexxSubtyp
                                                         .
                                                         .
                                                         .

/* Close the component. */
cmgr->CloseComponent(ev, sc);
```

CloseComponent - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

CountComponents

CountComponents - Syntax

This method returns the number of registered components that meet the search criteria.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>
```

```
ComponentDescription *plooking;
```

```
ODULong rc;

rc = CountComponents(plooking);
```

CountComponents Parameter - plooking

plooking ([ComponentDescription](#) *) - input

A component description record which specifies criteria for the component search. The Component Manager ignores fields in the component description record that are set to 0. For example, if all the fields are set to 0, the Component Manager returns the number of components registered in the system. Similarly, if all fields except the *componentManufacturer* field are set to 0, the Component Manager returns the number of registered components supplied by the specified manufacturer.

CountComponents Return Value - rc

rc ([ODULong](#)) - returns

The number of components that meet the search criteria.

CountComponents - Parameters

plooking ([ComponentDescription](#) *) - input

A component description record which specifies criteria for the component search. The Component Manager ignores fields in the component description record that are set to 0. For example, if all the fields are set to 0, the Component Manager returns the number of components registered in the system. Similarly, if all fields except the *componentManufacturer* field are set to 0, the Component Manager returns the number of registered components supplied by the specified manufacturer.

rc ([ODULong](#)) - returns

The number of components that meet the search criteria.

CountComponents - Example Code

This example return the number of components currently registered in the component registration data base.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription cd;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set all fields to zero in component description so that */
/* we count all installed components. */
cd.componentType = 0;
cd.componentSubType = 0;
```

```
cd.componentManufacturer = 0;
cd.componentFlags        = 0;
cd.componentFlagsMask     = 0;

printf("Components installed = %d\n", cmgr->CountComponents(ev, &cd));
```

CountComponents - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

FindNextComponent

FindNextComponent - Syntax

This method returns the component identifier of the registered component that meets the search criteria.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentDescription *plastFound;
ComponentDescription *plooking;
ComponentDescription pfound;
OSErr rc; /* Return code. */

rc = FindNextComponent(plastFound, plooking,
    pfound);
```

FindNextComponent Parameter - plastFound

plastFound ([ComponentDescription](#) *) - input

The starting point for the search. To start the search at the beginning of the component list, set this parameter to 0. To continue searching the remaining components, specify a component identifier previously returned by this method.

FindNextComponent Parameter - plooking

plooking ([ComponentDescription *](#)) - input

A component description record which specifies the search criteria.

The Component Manager ignores fields in the component description record that are set to 0. For example, if you set all fields to 0, all components meet the search criteria. In this case, the application can retrieve information about all of the components that are registered in the system by repeatedly calling `FindNextComponent` and [GetComponentInfo](#) methods until the search is complete. Similarly, if all the fields except the *componentManufacturer* field are set to 0, the Component Manager searches all registered components for a component supplied by the specified manufacturer.

FindNextComponent Parameter - pfound

pfound ([ComponentDescription](#)) - output

The component description that matches the search criteria.

FindNextComponent Return Value - rc

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errCMInvalidComponentID

This component is not found in the registration data base.

errCMSYSTEMError

The Component Manager encountered a system error.

FindNextComponent - Parameters

plastFound ([ComponentDescription *](#)) - input

The starting point for the search. To start the search at the beginning of the component list, set this parameter to 0. To continue searching the remaining components, specify a component identifier previously returned by this method.

plooking ([ComponentDescription *](#)) - input

A component description record which specifies the search criteria.

The Component Manager ignores fields in the component description record that are set to 0. For example, if you set all fields to 0, all components meet the search criteria. In this case, the application can retrieve information about all of the components that are registered in the system by repeatedly calling `FindNextComponent` and [GetComponentInfo](#) methods until the search is complete. Similarly, if all the fields except the *componentManufacturer* field are set to 0, the Component Manager searches all registered components for a component supplied by the specified manufacturer.

pfound ([ComponentDescription](#)) - output

The component description that matches the search criteria.

rc ([OSErr](#)) - returns

Return code.

noErr

No error.

errCMInvalidComponentID
This component is not found in the registration data base.

errCMSystemError
The Component Manager encountered a system error.

FindNextComponent - Remarks

To retrieve detailed information about a component, use the [GetComponentInfo](#) method to get the component description record for each returned component.

An application can use the component descriptor returned by [this method](#) to get more information about the component or to open the component.

FindNextComponent - Example Code

This example finds the next scripting component with a component type of **osa**.

```
Environment          *ev = somGetGlobalEnvironment();
ComponentManager     *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription  cd, theComponent;
OSErr                rc;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set component type to osa so we only search for scripting components. */
cd.componentType      = kOSAComponentType;    // 'osa '
cd.componentSubType   = 0;
cd.componentManufacturer = 0;
cd.componentFlags     = 0;
cd.componentFlagsMask = 0;

/* Find a scripting component. */
rc = cmgr->FindNextComponent(ev, NULL, &cd, &theComponent);
```

FindNextComponent - Topics

Class:
ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

GetComponentInfo

GetComponentInfo - Syntax

This method returns all of the attributes of the specified component.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentDescription *ptheComponent;
char *pcomponentName;
char *pcomponentInfo;
OSErr rc; /* Return code. */

rc = GetComponentInfo(ptheComponent, pcomponentName,
    pcomponentInfo);
```

GetComponentInfo Parameter - ptheComponent

ptheComponent ([ComponentDescription](#) *) - input

A component identifier that specifies the component whose information is to be returned. An application can obtain the component identifier from the [FindNextComponent](#) method.

GetComponentInfo Parameter - pcomponentName

pcomponentName (char *) - output

An existing character string buffer in which the component's name is to be returned. To ensure the string buffer is large enough, the buffer should be defined as a CMGRString type. If the component does not have a name, a null string is returned. This field can be set to NULL if you do not want to receive the component's name.

GetComponentInfo Parameter - pcomponentInfo

pcomponentInfo (char *) - output

An existing character string buffer in which the component's information string is to be returned. To ensure the string buffer is large enough, the buffer should be defined as a CMGRString type. If the component does not have an information string, a null string is returned. This parameter can be set to NULL if you do not want to receive the component's information string.

GetComponentInfo Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errCMInvalidComponentID	This component ID is not found in the registration data base.
errCMSystemError	The Component Manager encountered a system error.

GetComponentInfo - Parameters

ptheComponent ([ComponentDescription](#) *) - input

A component identifier that specifies the component whose information is to be returned. An application can obtain the component identifier from the [FindNextComponent](#) method.

pcomponentName (char *) - output

An existing character string buffer in which the component's name is to be returned. To ensure the string buffer is large enough, the buffer should be defined as a CMGRString type. If the component does not have a name, a null string is returned. This field can be set to NULL if you do not want to receive the component's name.

pcomponentInfo (char *) - output

An existing character string buffer in which the component's information string is to be returned. To ensure the string buffer is large enough, the buffer should be defined as a CMGRString type. If the component does not have an information string, a null string is returned. This parameter can be set to NULL if you do not want to receive the component's information string.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errCMInvalidComponentID	This component ID is not found in the registration data base.
errCMSystemError	The Component Manager encountered a system error.

GetComponentInfo - Example Code

This example finds the next scripting component with a component type of **osa** and returns the name and information of that component.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription cd, theComponent;
CMGRString        szComponentName;
CMGRString        szComponentInfo;
OSErr             rc;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set component type to osa so we only search for scripting components. */
cd.componentType = kOSAComponentType; /* 'osa ' */
```



```

cd.componentSubType      = 0;
cd.componentManufacturer = 0;
cd.componentFlags        = 0;
cd.componentFlagsMask    = 0;

/* find a scripting component. */
rc = cmgr->FindNextComponent(ev, NULL, &cd, &theComponent);

/* Get the name and information for the component we found. */
rc = cmgr->GetComponentInfo(ev, &theComponent, szComponentName, szComponent

```

GetComponentInfo - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

GetComponentRefcon

GetComponentRefcon - Syntax

This method returns the specified component's reference constant.

```

#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentDescription  *ptheComponent;
ODSLong              rc;

rc = GetComponentRefcon(ptheComponent);

```

GetComponentRefcon Parameter - ptheComponent

ptheComponent ([ComponentDescription](#) *) - input

The component whose reference constant is to be retrieved.

GetComponentRefcon Return Value - rc

rc ([ODSLong](#)) - returns
The reference constant of the component.

GetComponentRefcon - Parameters

theComponent ([ComponentDescription *](#)) - input
The component whose reference constant is to be retrieved.

rc ([ODSLong](#)) - returns
The reference constant of the component.

GetComponentRefcon - Example Code

This example finds the next scripting component with a component type of **osa** and returns the reference constant of that component.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription cd, theComponent;
ODSLong          refCon;
OSErr            rc;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set component type to osa so we only search for scripting components. */
cd.componentType      = kOSAComponentType; /* 'osa ' */
cd.componentSubType   = 0;
cd.componentManufacturer = 0;
cd.componentFlags     = 0;
cd.componentFlagsMask  = 0;

/* Find a scripting component. */
rc = cmgr->FindNextComponent(ev, NULL, &cd, &theComponent);

/* Get the refcon for the component we found. */
refCon = cmgr->GetComponentRefcon(ev, &theComponent);
```

GetComponentRefcon - Topics

Class:

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

InstallComponent (OS/2)

InstallComponent (OS/2) - Syntax

This method installs the component in the component registration data base.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentRegistryData    *ptheData;
OSErr                    rc;      /* Return code. */

rc = InstallComponent(ptheData);
```

InstallComponent (OS/2) Parameter - ptheData

ptheData ([ComponentRegistryData *](#)) - input
 A pointer to the component's registration information.

InstallComponent (OS/2) Return Value - rc

rc ([OSErr](#)) - returns
 Return code.

noErr	No error.
errCMComponentAlreadyInstalled	This component is already in the registration data base.
errCMBadParm	One or more of the parameters passed were invalid.
errCMSystemError	The Component Manager encountered a system error.

InstallComponent (OS/2) - Parameters

ptheData ([ComponentRegistryData *](#)) - input
A pointer to the component's registration information.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errCMComponentAlreadyInstalled	This component is already in the registration data base.
errCMBadParm	One or more of the parameters passed were invalid.
errCMSystemError	The Component Manager encountered a system error.

InstallComponent (OS/2) - Remarks

This method should be called only during the installation procedure for a component.

If a component with the same *componentType* and *componentSubType* is already registered, this method does not perform the registration and returns an error. In this case, version checking should be performed to ensure that the correct component level is being installed.

InstallComponent (OS/2) - Example Code

This example sets up the component registry information and adds the scripting component to the component registration data base.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
ComponentRegistryData crd;
OSErr            rc;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/*                                     */
/* Setup component registry information. */
/*                                     */

/* scripting component */
crd.componentType = kOSAComponentType;

/* subtype code for your component in big-endian format 'MYSC' --> 'CSYM' */
crd.componentSubType = 0x4353594D;

/* manufacturer code 'XYZ' */
crd.componentManufacturer = 0x58595A20;

/* flags indicating supported functions */
crd.componentFlags = 0x000001FE;

/* version of component */
crd.componentVersion = 1;
```

```

/* SOM Class name for component */
strcpy(crd.componentClassName, "REXXScriptingComponent");

/* DLL name for component without the '.dll' */
strcpy(crd.componentDLL, "rexxsc");

/* name of the component */
strcpy(crd.componentName, "Object REXX Scripting Component");

/* component Info */
strcpy(crd.componentInfo, "(C) IBM Corporation, 1994");

/* Install the component. */
rc = cmgr->InstallComponent(ev, &crd);

```

InstallComponent (OS/2) - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

OpenComponent

OpenComponent - Syntax

This method opens a connection to a component when the component type code and subtype code are known.

```

#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentDescription    *ptheComponent;
Component               *rc;

rc = OpenComponent(ptheComponent);

```

OpenComponent Parameter - ptheComponent

ptheComponent ([ComponentDescription *](#)) - input

The component description that specifies the component to be opened. The application can obtain this description from the [FindNextComponent](#) method.

OpenComponent Return Value - rc

rc ([Component *](#)) - returns

The component instance that identifies the application's connection to the component. A return code of NULL indicates that the specified component could not be opened.

OpenComponent - Parameters

ptheComponent ([ComponentDescription *](#)) - input

The component description that specifies the component to be opened. The application can obtain this description from the [FindNextComponent](#) method.

rc ([Component *](#)) - returns

The component instance that identifies the application's connection to the component. A return code of NULL indicates that the specified component could not be opened.

OpenComponent - Remarks

The component instance connects the application to the component and allows an application to gain access to the functions provided by the specified component. The application must open a component before it can call any component functions. The component is specified by a component identifier that the application previously obtained from the [FindNextComponent](#) [trom=textonly](#) method.

Alternatively, the [OpenDefaultComponent](#) method can be used to open a component without calling the [FindNextComponent](#) method.

Note: An application can maintain several connections to a single component, or it can have connections to several components at one time.

This component instance must be supplied whenever the functions provided by the component are called and when closing the component using the [CloseComponent](#) method.

OpenComponent - Related Methods

Related Methods

- [ComponentManager::FindNextComponent](#)
 - [ComponentManager::OpenDefaultComponent](#)
-

OpenComponent - Example Code

This example finds the next scripting component with a component type of **osa** and opens that component.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription cd, theComponent;
OSErr            rc;

/* create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set component type to osa so we only search for scripting component. */
cd.componentType      = kOSAComponentType;    // 'osa '
cd.componentSubType    = 0;
cd.componentManufacturer = 0;
cd.componentFlags      = 0;
cd.componentFlagsMask   = 0;

/* Find a scripting component. */
rc = cmgr->FindNextComponent(ev, NULL, &cd, &theComponent);

/* Open the component we just found. */
sc = (OSAScriptingComponent *) cmgr->OpenComponent(ev, &theComponent);
```

OpenComponent - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Methods](#)
[Glossary](#)

OpenDefaultComponent

OpenDefaultComponent - Syntax

This method returns the component instance that meets the search criteria.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
```

```
#include <os2.h>

OSType      componentType;
OSType      componentSubType;
Component   *rc;

rc = OpenDefaultComponent (componentType, componentSubType);
```

OpenDefaultComponent Parameter - componentType

componentType (OSType) - input

A four-character code that identifies the type of component. All components of a particular type support a common set of interface routines. An application uses this parameter to search for components of a give type.

OpenDefaultComponent Parameter - componentSubType

componentSubType (OSType) - input

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor.

This parameter can be set to 0 to select a component with any subtype value.

OpenDefaultComponent Return Value - rc

rc (Component *) - returns

The component that matches the search criteria. A return value of NULL indicates that the specified component could not be opened.

OpenDefaultComponent - Parameters

componentType (OSType) - input

A four-character code that identifies the type of component. All components of a particular type support a common set of interface routines. An application uses this parameter to search for components of a give type.

componentSubType (OSType) - input

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor.

This parameter can be set to 0 to select a component with any subtype value.

rc (Component *) - returns

The component that matches the search criteria. A return value of NULL indicates that the specified component could not be opened.

OpenDefaultComponent - Remarks

This method allows an application to gain access to the services provided by a component when the component type and subtype codes are known in advance. An application must open a component before it can call any component functions. The Component Manager searches for a component that meets the criteria set in the *componentType* and *componentSubType* parameters. If more control is needed over the selection process, the [FindNextComponent](#) method or the [OpenComponent](#) method can be used.

The OpenDefaultComponent method searches its list of registered components for a component that meets the search criteria. If a component that matches the search criteria is found, this method opens a connection to the component and returns a component instance. This returned component instance identifies the application's connection to the component. This component instance must be supplied whenever you call the functions provided by the component. To close the component, this component instance must also be supplied to the [CloseComponent](#) method.

If more than one component in the list of registered components meets the search criteria, OpenDefaultComponent opens the first one that it finds in the list.

OpenDefaultComponent - Related Methods

Related Methods

- [ComponentManager::FindNextComponent](#)
- [ComponentManager::OpenComponent](#)

OpenDefaultComponent - Example Code

This example opens a scripting component using the OpenDefaultComponent function.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
OSType           compType;
OSType           compSubType;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Open a scripting component */
compType = kOSAComponentType /* Want a scripting component. */
compSubType = 0;             /* Any scripting component will do. */
sc = (OSAScriptingComponent *) cmgr->OpenDefaultComponent(ev, compType,
                                                           compSubType);
```

OpenDefaultComponent - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)

SetComponentRefcon

SetComponentRefcon - Syntax

This method sets the reference constant for the component.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

ComponentDescription *ptheComponent;
ODSLong Refcon;

SetComponentRefcon(ptheComponent, Refcon);
```

SetComponentRefcon Parameter - ptheComponent

ptheComponent ([ComponentDescription](#) *) - input
The component description of the component whose reference constant is to be set.

SetComponentRefcon Parameter - Refcon

Refcon ([ODSLong](#)) - input
The reference constant value to be set.

SetComponentRefcon - Return Value

None.

SetComponentRefcon - Parameters

theComponent ([ComponentDescription](#) *) - input

The component description of the component whose reference constant is to be set.

Refcon ([ODSLong](#)) - input

The reference constant value to be set.

None.

SetComponentRefcon - Remarks

This method sets the value of the reference constant for a component. The component can later retrieve the reference constant using the [GetComponentRefcon](#) method.

SetComponentRefcon - Related Methods

Related Methods

- [ComponentManager::GetComponentRefcon](#)
-

SetComponentRefcon - Example Code

This example finds the next scripting component whose component type is **osa** and sets the reference constant for that component.

```
Environment      *ev = somGetGlobalEnvironment();
ComponentManager *cmgr;
OSAScriptingComponent *sc = NULL;
ComponentDescription cd, theComponent;
ODSLong           refCon;
OSErr             rc;

/* create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* Set component type to osa so we only search for scripting components. */
cd.componentType      = kOSAComponentType; /* 'osa' */
cd.componentSubType   = 0;
cd.componentManufacturer = 0;
cd.componentFlags     = 0;
cd.componentFlagsMask  = 0;

/* Find a scripting component. */
rc = cmgr->FindNextComponent(ev, NULL, &cd, &theComponent);
```

```
/* Set the refcon for the component we found. */
refCon = 9999;
cmgr->SetComponentRefcon(ev, &theComponent, refCon);
```

SetComponentRefcon - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Methods](#)

[Glossary](#)

UninstallComponent (OS/2)

UninstallComponent (OS/2) - Syntax

This method removes a component from the registration data base.

```
#define INCL_ODAPI
#define INCL_ODCOMPONENT
#include <os2.h>

OSType    componentType;
OSType    componentSubType;
OSErr     rc;          /* Return code. */

rc = UninstallComponent(componentType, componentSubType);
```

UninstallComponent (OS/2) Parameter - componentType

componentType ([OSType](#)) - input

The component type; for example, "osa" for the component to uninstall.

UninstallComponent (OS/2) Parameter - componentSubType

componentSubType (OSType) - input

The component subtype; for example, "OREX" for the component to uninstall.

UninstallComponent (OS/2) Return Value - rc

rc (OSErr) - returns

Return code.

noErr

No error.

errCMInvalidComponentID

This component ID is not found in the registration data base.

errCMSystemError

The Component Manager encountered a system error.

UninstallComponent (OS/2) - Parameters

componentType (OSType) - input

The component type; for example, "osa" for the component to uninstall.

componentSubType (OSType) - input

The component subtype; for example, "OREX" for the component to uninstall.

rc (OSErr) - returns

Return code.

noErr

No error.

errCMInvalidComponentID

This component ID is not found in the registration data base.

errCMSystemError

The Component Manager encountered a system error.

UninstallComponent (OS/2) - Example Code

This example removes the scripting component from the component registration data base whose component type is **osa** and whose subtype is **MYSC**.

```
Environment          *ev = somGetGlobalEnvironment();
ComponentManager     *cmgr;
OSErr                rc;
OSType               compType;
OSType               compSubType;

/* Create an instance of the Component Manager. */
cmgr = new ComponentManager;

/* scripting component */
compType = kOSAComponentType;
```

```
/* subtype code for your component in big-endian format 'MYSC' --> 'CSYM' *  
compSubType      = 0x4353594D;  
  
/* Uninstall the component. */  
rc = cmgr->UninstallComponent(ev, compType, compSubType);
```

UninstallComponent (OS/2) - Topics

Class:

ComponentManager

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

GenericScriptingComponent

Class Definition File: OSAGSC.IDL

Class Hierarchy

SOMObject
Component
OSAScriptingComponent
OSAGenericScriptingComponent

Description

Methods

The following list shows the methods defined by the OSAGenericScriptingComponent class:

- [OSAGenericToRealID](#)
- [OSAGetDefaultScriptingComponent](#)
- [OSAGetScriptingComponent](#)
- [OSAGetScriptingComponentFromStored](#)
- [OSARealToGenericID](#)
- [OSASetDefaultScriptingComponent](#)

Overridden Methods

There are currently no methods overridden by the OSAGenericScriptingComponent class.

OSAGenericToRealID

OSAGenericToRealID - Syntax

This method converts a generic script ID to the corresponding component-specific script ID.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      *ptheScriptID;
Component  **ptheExactComponent;
OSAError    rc;          /* Return code. */

rc = OSAGenericToRealID(ptheScriptID, ptheExactComponent);
```

OSAGenericToRealID Parameter - ptheScriptID

ptheScriptID (OSAID *) - in/out

The generic script ID to be converted. The component-specific script ID that corresponds to the generic script ID that you pass in this parameter.

OSAGenericToRealID Parameter - ptheExactComponent

ptheExactComponent (Component **) - output

The component instance that created the script returned in *ptheScriptID* .

OSAGenericToRealID Return Value - rc

rc (OSAError) - returns

Return code.

noErr

No error.

errOSACantOpenComponent

The scripting component cannot be connected.

errOSASystemError

A general scripting system error occurred

OSAGenericToRealID - Parameters

ptheScriptID (OSAID *) - in/out

The generic script ID to be converted. The component-specific script ID that corresponds to the generic script ID that you pass in this parameter.

ptheExactComponent (Component **) - output

The component instance that created the script returned in *ptheScriptID* .

rc ([OSAEError](#)) - returns
 Return code.

noErr
 No error.

errOSACantOpenComponent
 The scripting component cannot be connected.

errOSASystemError
 A general scripting system error occurred

OSAGenericToRealID - Remarks

You cannot use the generic scripting component and a generic script ID with component-specific routines. Instead, you can use the component instance and script ID returned by OSAGenericToRealID.

Given a generic script ID (that is, a script ID returned by a call to a standard component routine via the generic scripting component), the OSAGenericToRealID method returns the equivalent component-specific script ID and the component instance that created that script ID. The OSAGenericToRealID method modifies the script ID in place, changing the generic script ID you pass in the *ptheScriptID* parameter to the corresponding component-specific script ID.

OSAGenericToRealID - Topics

Class:
 OSAGenericScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

OSAGetDefaultScriptingComponent

OSAGetDefaultScriptingComponent - Syntax

This method returns the subtype code for the default scripting component associated with an instance of the generic scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ScriptingComponentSelector    *pscriptingSubType;
OSAError                      rc;          /* Return code. */

rc = OSAGetDefaultScriptingComponent (pscriptingSubType);
```

OSAGetDefaultScriptingComponent Parameter - pscribingSubType

pscribingSubType ([ScriptingComponentSelector](#) *) - output
The subtype code for the default scripting component.

OSAGetDefaultScriptingComponent Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.

OSAGetDefaultScriptingComponent - Parameters

pscribingSubType ([ScriptingComponentSelector](#) *) - output
The subtype code for the default scripting component.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.

OSAGetDefaultScriptingComponent - Remarks

This method returns the subtype code for the default scripting component. This is the scripting component that is used by [OSAStartRecording](#), [OSACompile](#), or [OSACompileExecute](#) if no existing script ID is specified. From the user's point of view, the default scripting component corresponds to the scripting language selected in the Script Editor application when the user first creates a new script.

Each instance of the generic scripting component has its own default scripting component, which is initially Object REXX. You can call the [OSASetDefaultScriptingComponent](#) method to change the default scripting component.

OSAGetDefaultScriptingComponent - Topics

Class:
[OSAGenericScriptingComponent](#)

Select an item:
[Syntax](#)

OSAGetScriptingComponent

OSAGetScriptingComponent - Syntax

This method returns an instance of the scripting component for a specified subtype.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ScriptingComponentSelector scriptingSubType;
Component **pscriptingInstance;
OSAError rc; /* Return code. */

rc = OSAGetScriptingComponent (scriptingSubType,
                               pscriptingInstance);
```

OSAGetScriptingComponent Parameter - scriptingSubType

scriptingSubType ([ScriptingComponentSelector](#)) - input
A subtype code for the scripting component.

OSAGetScriptingComponent Parameter - pscriptingInstance

pscriptingInstance (Component **) - output
A component instance for the scripting component identified by the *scriptingSubType* parameter.

OSAGetScriptingComponent Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr

	No error.
errOSACantOpenComponent	The scripting component cannot be connected.
errOSASystemError	A general scripting system error occurred.

OSAGetScriptingComponent - Parameters

scriptingSubType ([ScriptingComponentSelector](#)) - input
A subtype code for the scripting component.

pscriptingInstance (Component **) - output
A component instance for the scripting component identified by the *scriptingSubType* parameter.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantOpenComponent	The scripting component cannot be connected.
errOSASystemError	A general scripting system error occurred.

OSAGetScriptingComponent - Remarks

You cannot use the generic scripting component with component-specific routines. Instead, use an instance of the specific scripting component that can be obtained with OSAGetScriptingComponent.

This method returns, in the *pscriptingInstance* parameter, an instance of the scripting component identified by the parameter. Each instance of the generic scripting component keeps track of a single instance of each component subtype, so OSAGetScriptingComponent always returns the same instance of a specified scripting component that the generic scripting component uses for standard scripting component routines.

For example, you can use [OSAGetDefaultScriptingComponent](#) to get the subtype code for the default scripting component—that is, the scripting component used by the generic scripting component for new scripts. You can then get an instance of the default scripting component by passing its subtype code to OSAGetScriptingComponent. Finally, you could use the [GetComponentInfo](#) method of the Component Manager to obtain the name and display it to the user.

Similarly, you can pass kObjectREXXSubtype in the *scriptingSubType* parameter to obtain an instance of the Object REXX component.

OSAGetScriptingComponent - Topics

Class:
[OSAGenericScriptingComponent](#)

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

OSAGetScriptingComponentFromStored

OSAGetScriptingComponentFromStored - Syntax

This method returns the subtype code for a scripting component that created a storage descriptor record.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc          *pscriptData;
ScriptingComponentSelector *pscriptingSubType;
OSAError        rc;          /* Return code. */

rc = OSAGetScriptingComponentFromStored(pscriptData,
    pscriptingSubType);
```

OSAGetScriptingComponentFromStored Parameter - pscriptData

pscriptData ([AEDesc *](#)) - input

Either a generic storage descriptor record or a component-specific storage descriptor record.

OSAGetScriptingComponentFromStored Parameter - pscriptingSub

pscriptingSubType ([ScriptingComponentSelector *](#)) - output

A subtype code identifying the scripting component that created the descriptor record specified by the *pscriptData* parameter.

OSAGetScriptingComponentFromStored Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSAGetScriptingComponentFromStored - Parameters

pscriptData ([AEDesc *](#)) - input

Either a generic storage descriptor record or a component-specific storage descriptor record.

pscriptingSubType ([ScriptingComponentSelector *](#)) - output

A subtype code identifying the scripting component that created the descriptor record specified by the *pscriptData* parameter.

rc ([OSAEError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSAGetScriptingComponentFromStored - Remarks

This method returns, in the *pscriptingSubType* parameter, the subtype code for the scripting component that created the script data specified by the *pscriptData* parameter.

The generic scripting component automatically identifies the appropriate scripting component for you when you use it to call [OSALoad](#). By calling `OSAGetScriptingComponentFromStored`, you can determine, without loading a script, which scripting component created the script data.

OSAGetScriptingComponentFromStored - Topics

Class:

`OSAGenericScriptingComponent`

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

OSARRealToGenericID

OSARRealToGenericID - Syntax

This method converts a component-specific script ID to the corresponding generic script ID.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>
```

```
OSAID      *ptheScriptID;
Component  *theExactComponent;
```

```
OSAError rc; /* Return code. */  
rc = OSARealToGenericID(ptheScriptID, theExactComponent);
```

OSARealToGenericID Parameter - ptheScriptID

ptheScriptID (OSAID *) - in/out

On input, this parameter is the component-specific script ID that is to be converted. You must have obtained this script ID from the scripting component instance passed in the *theExactComponent* parameter.

On output, the generic script ID that corresponds to the component-specific script ID that you pass in this parameter is returned.

OSARealToGenericID Parameter - theExactComponent

theExactComponent (Component *) - input

A scripting component instance returned by a generic scripting component.

OSARealToGenericID Return Value - rc

rc (OSAError) - returns

Return code.

noErr

No error.

errOSAComponentMismatch

The *ptheScriptID* and *theExactComponent* parameters are for two different scripting components.

errOSASystemError

A general scripting system error occurred.

OSARealToGenericID - Parameters

ptheScriptID (OSAID *) - in/out

On input, this parameter is the component-specific script ID that is to be converted. You must have obtained this script ID from the scripting component instance passed in the *theExactComponent* parameter.

On output, the generic script ID that corresponds to the component-specific script ID that you pass in this parameter is returned.

theExactComponent (Component *) - input

A scripting component instance returned by a generic scripting component.

rc (OSAError) - returns

Return code.

noErr

No error.

errOSAComponentMismatch

The *ptheScriptID* and *theExactComponent* parameters are for two different scripting components.

errOSASystemError

A general scripting system error occurred.

OSASRealToGenericID - Remarks

This method performs the reverse task performed by [OSAGenericToRealID](#). Given a component-specific script ID and an exact scripting component instance (that is, the component instance that created the component-specific script ID), the OSASRealToGenericID method returns the corresponding generic script ID. The OSASRealToGenericID method modifies the script ID in place, changing the component-specific script ID passed in the *ptheScriptID* parameter to the corresponding generic script ID.

You will need to do this if you have obtained a component-specific script ID using an exact scripting component instance and you want to refer to the same script in calls that use an instance of the generic scripting component. You cannot use a component-specific script ID with the generic scripting component.

The script ID you pass in the *ptheScriptID* parameter must be a component-specific script ID obtained from a scripting component instance known to the generic scripting component. You can obtain such an instance by calling either [OSAGetScriptingComponent](#) or [OSAGenericToRealID](#).

OSASRealToGenericID - Topics

Class:

[OSAGenericScriptingComponent](#)

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

OSASetDefaultScriptingComponent

OSASetDefaultScriptingComponent - Syntax

This method sets the default scripting component associated with an instance of the generic scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ScriptingComponentSelector scriptingSubType;
OSAError rc; /* Return code. */

rc = OSASetDefaultScriptingComponent (scriptingSubType);
```

OSASetDefaultScriptingComponent Parameter - scriptingSubType

scriptingSubType ([ScriptingComponentSelector](#)) - input
The subtype code for the scripting component you want to set as the default.

OSASetDefaultScriptingComponent Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantOpenComponent	The specified component subtype has not been registered.
errOSASystemError	A general scripting system error occurred.

OSASetDefaultScriptingComponent - Parameters

scriptingSubType ([ScriptingComponentSelector](#)) - input
The subtype code for the scripting component you want to set as the default.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantOpenComponent	The specified component subtype has not been registered.
errOSASystemError	A general scripting system error occurred.

OSASetDefaultScriptingComponent - Remarks

This method sets the default scripting component for the specified instance of the generic scripting component to the scripting component identified by the *scriptingSubType* parameter.

Each instance of the generic scripting component has its own default scripting component, which is initially Object REXX. You can use [OSAGetDefaultScriptingComponent](#) to get the current default scripting component for an instance of the generic scripting component.

OSASetDefaultScriptingComponent - Topics

Class:

OSAGenericScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

OSAScriptingComponent

Class Definition File: OSASC.IDL

Class Hierarchy

SOMObject

Component

OSAScriptingComponent

Description

The OSAScriptingComponent class is an abstract base class for scripting components. Only the methods for manipulating trailers for generic storage descriptor records have an implementation: [OSAGetStorageType](#), [OSAAddStorageType](#), and [OSARemoveStorageType](#). Scripting components should not override these methods.

Methods that depend on the capabilities of scripting components (for example, dialect support) have a stub implementation that returns the error `errOSAMessageNotUnderstood` if a scripting component does not register support of a capability in its *componentFlags* ; it does not need to subclass these methods. All methods that a scripting component does support must be overridden.

A *script context* maintains context information for the execution of other scripts. A script context can also contain executable statements in a scripting language. Like a compile script, a script context can be decompiled into source statements.

An application specifies an optional script context when it executes a script. A script can be associated with an OSA event object (described in an object specifier record) in the form of a script context. It is then executed when a specified OSA event performs an action on that object.

Scripting components can bind variables with the aid of scripting contexts. They can maintain the state information for a script object in the script context. Because script contexts can be stored and loaded just like any other script, they also provide a means for achieving persistence.

Script contexts can also contain handlers for OSA events which an application can invoke by calling [OSADoEvent](#) and [OSAExecuteEvent](#) methods.

The methods defined in this class are used by applications to manipulate and execute scripts written in an OSA compatible scripting language. [Introduction to Scripting](#) introduces how applications and part handlers implement scripting support for objects within embedded scripts.

Methods

The following list shows the methods defined by the OSAScriptingComponent class:

- [OSAAddStorageType](#)
- [OSAAvailableDialectCodeList](#)
- [OSAAvailableDialects](#)
- [OSACoerceFromDesc](#)
- [OSACoerceToDesc](#)
- [OSACompile](#)
- [OSACompileExecute](#)
- [OSACopyID](#)
- [OSADisplay](#)
- [OSADispose](#)
- [OSADoEvent](#)
- [OSADoScript](#)
- [OSAExecute](#)
- [OSAExecuteEvent](#)
- [OSAGetActiveProc](#)

- [OSAGetCreateProc](#)
- [OSAGetCurrentDialect](#)
- [OSAGetDialectInfo](#)
- [OSAGetResumeDispatchProc](#)
- [OSAGetScriptInfo](#)
- [OSAGetSendProc](#)
- [OSAGetSource](#)
- [OSAGetStorageType](#)
- [OSALoad](#)
- [OSALoadExecute](#)
- [OSAMakeContext](#)
- [OSARemoveStorageType](#)
- [OSAScriptError](#)
- [OSAScriptingComponentName](#)
- [OSASetActiveProc](#)
- [OSASetCreateProc](#)
- [OSASetCurrentDialect](#)
- [OSASetDefaultTarget](#)
- [OSASetScriptInfo](#)
- [OSASetSendProc](#)
- [OSASetResumeDispatchProc](#)
- [OSAStartRecording](#)
- [OSAStopRecording](#)
- [OSAStore](#)

Overridden Methods

There are currently no methods overridden by the OSAScriptingComponent class.

OSAAddStorageType

OSAAddStorageType - Syntax

This method adds a trailer to the script data in a generic storage descriptor record.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ODUByte    *oldScriptData;
ODULong    oldSize;
ODUByte    *newScriptData;
ODULong    *newSize;
DescType    dtype;
OSErr      rc;          /* Return code. */

rc = OSAAddStorageType(oldScriptData, oldSize,
                       newScriptData, newSize, dtype);
```

OSAAddStorageType Parameter - oldScriptData

oldScriptData (ODUByte *) - input
A pointer to the script data.

OSAAddStorageType Parameter - oldSize

oldSize (ODULong) - input
The size of the old script data.

OSAAddStorageType Parameter - newScriptData

newScriptData (ODUByte *) - output
A pointer to memory allocated by the user to hold the script data plus the storage type. If this parameter is passed as NULL, the required size is returned in the *newSize* parameter.

OSAAddStorageType Parameter - newSize

newSize (ODULong *) - in/out
The size of the new script data.

OSAAddStorageType Parameter - dtype

dtype (DescType) - input
The descriptor type to be specified in the trailer added to the script data.

OSAAddStorageType Return Value - rc

rc (OSError) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSABufferTooSmall	The size of the buffer is not large enough to hold the data to be returned.

OSAAddStorageType - Parameters

oldScriptData ([ODUByte *](#)) - input
A pointer to the script data.

oldSize ([ODULong](#)) - input
The size of the old script data.

newScriptData ([ODUByte *](#)) - output
A pointer to memory allocated by the user to hold the script data plus the storage type. If this parameter is passed as NULL, the required size is returned in the *newSize* parameter.

newSize ([ODULong *](#)) - in/out
The size of the new script data.

dtype ([DescType](#)) - input
The descriptor type to be specified in the trailer added to the script data.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSABufferTooSmall	The size of the buffer is not large enough to hold the data to be returned.

OSAAddStorageType - Remarks

This method attaches a trailer to a handle (consequently expanding the data to which the handle refers) or updates an existing trailer.

A scripting component does not have to instantiate an instance of the [GenericScriptingComponent](#) to use this method.

OSAAddStorageType - Override Policy

A scripting component should not override this method.

OSAAddStorageType - Topics

Class:
[OSAScriptingComponent](#)

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSAAvailableDialectCodeList

OSAAvailableDialectCodeList - Syntax

This method obtains a descriptor list containing dialect codes for each of a scripting component's currently available dialects.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *presultingDialectCodeList;
OSAError    rc;                                /* Return code. */

rc = OSAAvailableDialectCodeList (presultingDialectCodeList);
```

OSAAvailableDialectCodeList Parameter - presultingDialectCodeLis

presultingDialectCodeList ([AEDesc *](#)) - output
The returned descriptor list.

OSAAvailableDialectCodeList Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.

OSAAvailableDialectCodeList - Parameters

presultingDialectCodeList ([AEDesc *](#)) - output
The returned descriptor list.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.

OSAAvailableDialectCodeList - Remarks

Each item in the descriptor list returned by this method is a descriptor record of descriptor type typeInteger containing a dialect code for one of the specified scripting component's currently available dialects. Dialect codes are defined by individual scripting components.

You can pass any dialect code obtained using OSAAvailableDialectCodeList to [OSAGetDialectInfo](#) to get information about the corresponding dialect.

OSAAvailableDialectCodeList - Override Policy

A scripting component must override this method if dialects are supported.

OSAAvailableDialectCodeList - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSAAvailableDialects

OSAAvailableDialects - Syntax

This method obtains a descriptor list containing information about each of the currently available dialects for a scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *presultingDialectInfoList;
OSAError    rc;                                /* Return code. */

rc = OSAAvailableDialects(presultingDialectInfoList);
```

OSAAvailableDialects Parameter - presultingDialectInfoList

resultingDialectInfoList ([AEDesc *](#)) - output
The returned descriptor list.

OSAAvailableDialects Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr No error.
errOSASystemError A general scripting system error occurred.

OSAAvailableDialects - Parameters

resultingDialectInfoList ([AEDesc *](#)) - output
The returned descriptor list.

rc ([OSAError](#)) - returns
Return code.

noErr No error.
errOSASystemError A general scripting system error occurred.

OSAAvailableDialects - Remarks

Each item in the list returned by this method is an AE record of descriptor type `typeOSADialectInfo`.

```
#define typeOSADialectInfo                      0x6F666964            /* "difo" */
```

Each descriptor record in the descriptor list contains, at a minimum, four keyword-specified descriptor records with the following keywords:

```
#define keyOSADialectName                      0x6D616E64            /* "dnam" */  
#define keyOSADialectCode                    0x646F6364            /* "dcod" */  
#define keyOSADialectLangCode                0x64636C64            /* "dlcd" */  
#define keyOSADialectScriptCode              0x64637364            /* "dscd" */
```

Rather than calling `OSAAvailableDialects` to obtain complete dialect information for a scripting component, it is usually more convenient to call [OSAAvailableDialectCodeList](#) to get a list of codes for a scripting component's dialects and then call [OSAGetDialectInfo](#) to get information about the specific dialect you are interested in.

OSAAvailableDialects - Override Policy

A scripting component must override this method if dialects are supported.

OSAAvailableDialects - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSACoerceFromDesc

OSACoerceFromDesc - Syntax

This method obtains the script ID for a script value that corresponds to the data in a descriptor record.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *pscriptData;
ODSLong     modeFlags;
OSAID       *presultingScriptID;
OSAError     rc;          /* Return code. */

rc = OSACoerceFromDesc(pscriptData, modeFlags,
                       presultingScriptID);
```

OSACoerceFromDesc Parameter - pscriptData

pscriptData ([AEDesc *](#)) - input

A descriptor record containing the script data to be coerced.

OSACoerceFromDesc Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. If the *pscriptData* parameter contains an OSA event, this parameter can be set to any of the mode flags listed in the following list:

kOSANullScript

The scripting component should use its default context.

kOSAModePreventGetSource

Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to [OSAGetSource](#).

kOSACompileIntoContext

The [OSACoerceFromDesc](#) method returns a script context instead of a compiled script.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSACoerceFromDesc Parameter - presulatingScriptID

presulatingScriptID ([OSAID](#) *) - output

The resulting script ID for a script value.

OSACoerceFromDesc Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSACoerceFromDesc - Parameters

pscriptData ([AEDesc](#) *) - input

A descriptor record containing the script data to be coerced.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. If the *pscriptData* parameter contains an OSA event, this parameter can be set to any of the mode flags listed in the following list:

kOSANullScript

The scripting component should use its default context.

kOSAModePreventGetSource

Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to [OSAGetSource](#).

kOSACompileIntoContext

The *OSACoerceFromDesc* method returns a script context instead of a compiled script.

kOSAModeNeverInteract

Adds *kAENeverInteract* to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds *kAECanInteract* to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds *kAEAlwaysInteract* to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDontReconnect

Adds *kAEDontReconnect* to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of *kAECanSwitchLayer* in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDoRecord

Prevents use of *kAEDontRecord* in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

resultingScriptID ([OSAID](#) *) - output

The resulting script ID for a script value.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSACoerceFromDesc - Remarks

This method coerces the descriptor record in the *pscriptData* parameter to the equivalent script value and returns a script ID for that value.

If you pass *OSACoerceFromDesc* an OSA event in the *pscriptData* parameter, it returns a script ID for the equivalent compiled script in the *resultingScriptID* parameter. In this case, you can specify any of the *modeFlags* values used by [OSACompile](#) to control the way the compiled script is executed.

Special Considerations

If you call *OSACoerceFromDesc* using an instance of the generic scripting component, the generic scripting component uses the default scripting component to perform the coercion.

For more information about the default scripting component associated with any instance of the generic scripting component, see [Generic Scripting Component Routines](#).

OSACoerceFromDesc - Override Policy

A scripting component which sets *kOSASupportsAECOercion* in the *componentFlags* field of the component description record must override

this method.

OSACoerceFromDesc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSACoerceToDesc

OSACoerceToDesc - Syntax

This method coerces a script value to a descriptor record of a desired descriptor type.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptID;
DescType   desiredType;
ODSLong    modeFlags;
AEDesc     *presult;
OSAError   rc;          /* Return codes. */

rc = OSACoerceToDesc(scriptID, desiredType,
                     modeFlags, presult);
```

OSACoerceToDesc Parameter - scriptID

scriptID ([OSAID](#)) - input

The script ID for the script value to coerce.

OSACoerceToDesc Parameter - desiredType

desiredType ([DescType](#)) - input
The desired descriptor type of the resulting descriptor record.

OSACoerceToDesc Parameter - modeFlags

modeFlags ([ODSLong](#)) - input
Information used by individual scripting components. This parameter can be set to one of the following mode flags:

- kOSAModeNull**
No mode flags are set.
 - kOSAModeNeverInteract**
Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.
 - kOSAModeCanInteract**
Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.
 - kOSAModeAlwaysInteract**
Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.
 - kOSAModeCantSwitchLayer**
Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).
 - kOSAModeDontReconnect**
Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.
 - kOSAModeDoRecord**
Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).
-

OSACoerceToDesc Parameter - presult

presult ([AEDesc *](#)) - output
The resulting descriptor record.

OSACoerceToDesc Return Value - rc

rc ([OSAEError](#)) - returns
Return codes.

- noErr**
No error.
 - errOSASystemError**
A general scripting system error occurred.
 - errOSAInvalidID**
The specified script ID is invalid.
-

OSACoerceToDesc - Parameters

scriptID ([OSAID](#)) - input

The script ID for the script value to coerce.

desiredType ([DescType](#)) - input

The desired descriptor type of the resulting descriptor record.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

result ([AEDesc *](#)) - output

The resulting descriptor record.

rc ([OSAError](#)) - returns

Return codes.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

OSACoerceToDesc - Remarks

This method coerces the script value identified by *scriptID* to a descriptor record of the type specified by the *desiredType* parameter, if possible. Valid types include all the standard descriptor types defined in the *OSA Event Registry: Standard Suites*, plus any special types supported by the scripting component.

Special Considerations

If you want the descriptor type of the descriptor record returned in the *result* parameter to be the same as the descriptor type returned by a scripting component, use `OSACoerceToDesc` and specify `typeWildcard` as the desired type. If you want to get a script value in a form that you can display for users to read, use [OSADisplay](#).

OSACoerceToDesc - Override Policy

A scripting component which sets `kOSASupportsAECOercion` in the *componentFlags* field of the component description record must override this method.

OSACoerceToDesc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSACompile

OSACompile - Syntax

This method compiles the source data for a script and returns the script ID for a compiled script or script context.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *psourceData;
ODSLong     modeFlags;
OSAID       *ppreviousAndResultingScriptID;
OSAError    rc;                                /* Return code. */

rc = OSACompile(psourceData, modeFlags, ppreviousAndResultingScriptID);
```

OSACompile Parameter - psourceData

psourceData ([AEDesc](#) *) - input

A descriptor record containing suitable source data for the specified scripting component.

OSACompile Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull	No mode flags are set.
kOSAModePreventGetSource	Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to OSAGetSource.
kOSACompileIntoContext	The OSACompile function returns a script context instead of a compiled script.
kOSAModeAugmentContext	Script data associated with script ID passed in <i>ppreviousAndResultingScriptID</i> parameter is augmented rather than replaced with the new compiled script. Specifying this flag automatically invokes the kOSAModeCompileIntoContext mode flag. If you redefine variables, handlers, and so on that were previously defined in the script context, the new definitions will replace the old ones.
kOSAModeNeverInteract	Adds kAENeverInteract to <i>sendMode</i> parameter of AESend for events sent when script is executed.
kOSAModeCanInteract	Adds kAECanInteract to <i>sendMode</i> parameter of AESend for events sent when script is executed.
kOSAModeAlwaysInteract	Adds kAEAlwaysInteract to <i>sendMode</i> parameter of AESend for events sent when script is executed.
kOSAModeDontReconnect	Adds kAEDontReconnect to <i>sendMode</i> parameter of AESend for events sent when script is executed.
kOSAModeCantSwitchLayer	Prevents use of kAECanSwitchLayer in <i>sendMode</i> parameter of AESend for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).
kOSAModeDoRecord	Prevents use of kAEDontRecord in <i>sendMode</i> parameter of AESend for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSACompile Parameter - ppreviousAndResultingScriptID

ppreviousAndResultingScriptID ([OSAID](#) *) - in/out

The script ID for the resulting compiled script. If the value of this parameter on input is kOSANullScript, this function returns a new script ID for the compiled script data. If the value of this parameter on input is an existing script ID, this function updates the script ID so that it refers to the newly compiled script data.

OSACompile Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantCoerce	The source data is incompatible with the scripting component.
errOSASystemError	A general scripting system error occurred.
errOSAInvalidID	The specified script ID is invalid.
errOSAScriptError	The source data is invalid (syntax error).

OSACompile - Parameters

psourceData ([AEDesc *](#)) - input

A descriptor record containing suitable source data for the specified scripting component.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModePreventGetSource

Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to OSAGetSource.

kOSACompileIntoContext

The OSACompile function returns a script context instead of a compiled script.

kOSAModeAugmentContext

Script data associated with script ID passed in *ppreviousAndResultingScriptID* parameter is augmented rather than replaced with the new compiled script. Specifying this flag automatically invokes the kOSAModeCompileIntoContext mode flag. If you redefine variables, handlers, and so on that were previously defined in the script context, the new definitions will replace the old ones.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

ppreviousAndResultingScriptID ([OSAID *](#)) - in/out

The script ID for the resulting compiled script. If the value of this parameter on input is kOSANullScript, this function returns a new script ID for the compiled script data. If the value of this parameter on input is an existing script ID, this function updates the script ID so that it refers to the newly compiled script data.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSACantCoerce

The source data is incompatible with the scripting component.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

The source data is invalid (syntax error).

OSACompile - Remarks

You can pass a descriptor record containing source data suitable for a specific scripting component (usually text) to the OSACompile function to obtain a script ID for the equivalent compiled script or script context. To compile the source data as a script context for use with [OSAExecuteEvent](#) or [OSADoEvent](#), you must set the kOSACompileIntoContext flag, and the source data should include appropriate handlers.

After you have successfully compiled the script, you can use the returned script ID to refer to the compiled script when you call [OSAExecute](#) and other scripting component routines.

Special Considerations

If you use this function with an instance of the generic scripting component and pass kOSANullScript in the *ppreviousAndResultingScriptID* parameter, the generic scripting component uses the default scripting component to compile the script.

If you are recompiling a script, specify the original script ID in the *ppreviousAndResultingScriptID* parameter. The generic scripting component uses the script ID to determine which scripting component it should use to compile the script.

For more information about the default scripting component associated with any instance of the generic scripting component, see [Generic Scripting Component Routines](#).

Object REXX Notes

The *psourceData* parameter must be of type typeChar.

OSACompile - Override Policy

A scripting component which sets kOSASupportsCompiling in the *componentFlags* field of the component description record must override this method.

OSACompile - Example Code

For an example of the use of the OSACompile method to update an existing script ID, see the figure in section [Modifying and Recompiling a Compiled Script](#). For an example of the use of OSACompile method to obtain a new script ID, see the figure in section [Compiling and Executing Source Data](#).

OSACompile - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Example Code](#)
[Glossary](#)

OSACompileExecute

OSACompileExecute - Syntax

This method compiles and executes a script in a single step rather than calling [OSACompile](#) and [OSAExecute](#).

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *sourceData;
OSAID       contextID;
ODSLong     modeFlags;
OSAID       *presultingScriptValueID;
OSAError    rc;                /* Return code. */

rc = OSACompileExecute(sourceData, contextID,
                      modeFlags, presultingScriptValueID);
```

OSACompileExecute Parameter - sourceData

sourceData ([AEDesc *](#)) - input

A descriptor record identifying suitable source data for the specified scripting component.

OSACompileExecute Parameter - contextID

contextID ([OSAID](#)) - input

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context.

OSACompileExecute Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. Other possible mode flags are described in the following list:

`kOSAModeNull`

No mode flags are set.

`kOSAModeNeverInteract`

Adds `kAENeverInteract` to *sendMode* parameter of [AESend](#) for events sent when script is executed.

`kOSAModeCanInteract`

Adds `kAECanInteract` to *sendMode* parameter of [AESend](#) for events sent when script is executed.

`kOSAModeAlwaysInteract`

Adds `kAEAAlwaysInteract` to *sendMode* parameter of [AESend](#) for events sent when script is executed.

`kOSAModeCantSwitchLayer`

Prevents use of `kAECanSwitchLayer` in *sendMode* parameter of [AESend](#) for events sent when script is executed

(the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSACompileExecute Parameter - presulatingScriptValueID

presulatingScriptValueID ([OSAID](#) *) - output

The script ID for the script value returned.

OSACompileExecute Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error occurred.

errOSACantCoerce

The data could not be coerced to the requested data type.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

The source data is invalid (syntax error), or an execution error occurred.

OSACompileExecute - Parameters

sourceData ([AEDesc](#) *) - input

A descriptor record identifying suitable source data for the specified scripting component.

contextID ([OSAID](#)) - input

The script ID for the context to be used during script execution. The constant kOSANullScript in this parameter indicates that the scripting component should use its default context.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. Other possible mode flags are described in the following list:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

resultingScriptValueID ([OSAID](#) *) - output

The script ID for the script value returned.

rc ([OSAError](#)) - returns

Return code.

noErr

No error occurred.

errOSACantCoerce

The data could not be coerced to the requested data type.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

The source data is invalid (syntax error), or an execution error occurred.

OSACompileExecute - Remarks

This method compiles source data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, OSACompileExecute disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant kOSANullScript.

If the result code returned by OSACompileExecute is a general result code, there was some problem in arranging for the script to be run. If the result code is errOSAScriptError, an error occurred during script execution. In this case, you can obtain more detailed error information by calling [OSAScriptError](#).

OSACompileExecute - Override Policy

A scripting component which sets kOSASupportsConvenience in the *componentFlags* field of the component description record must override this method.

OSACompileExecute - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSACopyID

OSACopyID - Syntax

This method updates script data after editing or recording and performs undo or revert operations on script data.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      fromID;
OSAID      *ptoID;
OSAError    rc;      /* Return code. */

rc = OSACopyID(fromID, ptoID);
```

OSACopyID Parameter - fromID

fromID (OSAID) - input

The script ID of the script data to be associated with the script ID in the *ptoID* parameter.

OSACopyID Parameter - ptoID

ptoID (OSAID *) - output

The script ID of the script data to be replaced or kOSANullScript if a new script ID is returned.

OSACopyID Return Value - rc

rc (OSAError) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

OSACopyID - Parameters

fromID ([OSAID](#)) - input

The script ID of the script data to be associated with the script ID in the *ptoid* parameter.

ptoid ([OSAID *](#)) - output

The script ID of the script data to be replaced or kOSANullScript if a new script ID is returned.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

OSACopyID - Remarks

This method replaces the script data identified by the script ID in the *ptoid* parameter with the script data identified by the script ID in the *fromID* parameter.

Object REXX Notes

The *ptoid* parameter must be kOSANullScript.

OSACopyID - Override Policy

A scripting component which sets kOSASupportsCompiling in the *componentFlags* field of the component description record must override this method.

OSACopyID - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSADisplay

OSADisplay - Syntax

This method converts a script value to text. The application can then use its own routines to display this text to the user.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptValueID;
DescType   desiredType;
ODULong    modeFlags;
AEDesc     *presultingText;
OSAError    rc;          /* Return code. */

rc = OSADisplay(scriptValueID, desiredType,
                modeFlags, presultingText);
```

OSADisplay Parameter - scriptValueID

scriptValueID ([OSAID](#)) - input
The script ID for the script value to coerce.

OSADisplay Parameter - desiredType

desiredType ([DescType](#)) - input
The desired text descriptor type, such as typeChar, for the resulting descriptor record.

OSADisplay Parameter - modeFlags

modeFlags ([ODULong](#)) - input
Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull
No mode flags are set.

kOSAModeDisplayForHumans
The resulting text is readable only by humans and cannot be recompiled.

OSADisplay Parameter - presultingText

presultingText ([AEDesc *](#)) - output
The resulting descriptor record.

OSADisplay Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantCoerce	The desired type is not supported by scripting component.
errOSASystemError	A general scripting system error occurred.
errOSAInvalidID	The specified script ID is invalid.

OSADisplay - Parameters

scriptValueID ([OSAID](#)) - input
The script ID for the script value to coerce.

desiredType ([DescType](#)) - input
The desired text descriptor type, such as typeChar, for the resulting descriptor record.

modeFlags ([ODULong](#)) - input
Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull	No mode flags are set.
kOSAModeDisplayForHumans	The resulting text is readable only by humans and cannot be recompiled.

resultingText ([AEDesc *](#)) - output
The resulting descriptor record.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantCoerce	The desired type is not supported by scripting component.
errOSASystemError	A general scripting system error occurred.
errOSAInvalidID	The specified script ID is invalid.

OSADisplay - Remarks

This function coerces the script value identified by *scriptValueID* to a descriptor record of the text type specified by the *desiredType* parameter, if possible. Valid types include all the standard text descriptor types defined in the *OSA Event Registry: Standard Suites*, plus any special types supported by the scripting component. Unlike [OSAGetSource](#), OSADisplay can coerce only script values and always produces a descriptor record of a text descriptor type.

Special Considerations

The OSADisplay method can be used to get a script value in a form that can be displayed for humans to read. If you want the descriptor type

of the descriptor record returned in the *presultingText* parameter to be the same as the descriptor type returned by a scripting component, use the [OSACoerceToDesc](#) method and specify typeWildcard as the desired type.

Object REXX Notes

The *desiredType* parameter must be of type typeChar, and the *modeFlags* parameter must be kOSANullScript.

OSADisplay - Override Policy

A scripting component must override this method.

OSADisplay - Example Code

For an example of the use of OSADisplay, see the figure in section [Compiling and Executing Source Data](#).

OSADisplay - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Example Code](#)
[Glossary](#)

OSADispose

OSADispose - Syntax

This method reclaims the memory occupied by script data.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptID;
OSAError    rc;      /* Return code. */

rc = OSADispose(scriptID);
```

OSADispose Parameter - scriptID

scriptID ([OSAID](#)) - input
The script ID of the script data to be disposed.

OSADispose Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSAInvalidID	The specified script ID is invalid.

OSADispose - Parameters

scriptID ([OSAID](#)) - input
The script ID of the script data to be disposed.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSAInvalidID	The specified script ID is invalid.

OSADispose - Remarks

This method releases the memory assigned to the script data identified by the *scriptID* parameter. The script ID passed to this method is no longer valid if the method returns successfully. A scripting component can then reuse that script ID for other script data.

A call to OSADispose returns noErr if the *scriptID* is kOSANullScript, although it does not dispose of anything.

OSADispose - Override Policy

A scripting component must override this method.

OSADispose - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSADoEvent

OSADoEvent - Syntax

This method handles an OSA event with the aid of a script context and obtains a reply event.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAEvent      *ptheOSAEvent;
OSAID         contextID;
ODSLong       modeFlags;
OSAEvent      *preply;
OSAError      rc;          /* Return code. */

rc = OSADoEvent(ptheOSAEvent, contextID, modeFlags,
               preply);
```

OSADoEvent Parameter - ptheOSAEvent

ptheOSAEvent ([OSAEvent](#) *) - input
The OSA event to be handled.

OSADoEvent Parameter - contextID

contextID ([OSAID](#)) - input
The script ID for the script context to be used to handle the OSA event.

OSADoEvent Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to any of the following flags:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSADoEvent Parameter - preply

preply ([OSAEvent *](#)) - in/out

The reply OSA event.

OSADoEvent Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errAEEEventNotHandled

The script context does not contain handler for event.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

OSADoEvent - Parameters

ptheOSAEvent ([OSAEvent *](#)) - input
The OSA event to be handled.

contextID ([OSAID](#)) - input
The script ID for the script context to be used to handle the OSA event.

modeFlags ([ODSLong](#)) - input
Information used by individual scripting components. This parameter can be set to any of the following flags:

kOSAModeNull
No mode flags are set.

kOSAModeNeverInteract
Adds **kAENeverInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract
Adds **kAECanInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract
Adds **kAEAlwaysInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer
Prevents use of **kAECanSwitchLayer** in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect
Adds **kAEDontReconnect** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord
Prevents use of **kAEDontRecord** in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

preply ([OSAEvent *](#)) - in/out
The reply OSA event.

rc ([OSAError](#)) - returns
Return code.

noErr
No error.

errAEEEventNotHandled
The script context does not contain handler for event.

errOSASystemError
A general scripting system error occurred.

errOSAInvalidID
The specified script ID is invalid.

OSADoEvent - Remarks

This method resembles both [OSADoScript](#) and [OSAExecuteEvent](#); however, unlike [OSADoScript](#), the script [OSADoEvent](#) executes must be in the form of a script context, and execution is initiated by an OSA event. Unlike [OSAExecuteEvent from=textonly](#), [OSADoEvent](#) returns a reply OSA event rather than the script ID of the resulting script value.

The [OSADoEvent](#) function, like [OSAExecuteEvent](#), attempts to use the script context specified by the *contextID* parameter to handle the OSA event specified by the *ptheOSAEvent* parameter. If the scripting component determines that the script context cannot handle the event (for example, if a script written in an Object REXX dialect does not include statements that handle the event), [OSADoEvent](#) immediately returns **errAEEEventNotHandled** rather than **errOSAScriptError**.

If the scripting component determines that the script context can handle the event, [OSADoEvent](#) executes the script context's handler for the event and returns the resulting script ID.

This method returns a reply event that contains either the resulting script value or, if an error occurred during script execution, information about the error. If the error **errOSAScriptError** occurs during script execution, [OSADoEvent](#) calls [OSAScriptError](#) and returns the appropriate error information in the reply. The [OSADoEvent](#) method never returns **errOSAScriptError**.

If the script context specifies that the OSA event should be passed to the application's standard handler for that event (for example, with an

Object REXX continue statement), OSADoEvent passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event (that is, without calling OSADoEvent again). If the contextID parameter is kOSANullScript, the OSADoEvent method passes the event directly to the resume dispatch function. If the call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Note: Like [OSAExecuteEvent](#), OSADoEvent can generate the result code errAEEEventNotHandled in at least two ways. If the scripting component determines that a script context does not declare a handler for a particular event, OSADoEvent immediately returns errAEEEventNotHandled. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, OSADoEvent returns errAEEEventNotHandled in the reply OSA event.

OSADoEvent - Override Policy

A scripting component which sets kOSASupportsEventHandling in the *componentFlags* field of the component description record must override this method.

OSADoEvent - Example Code

For an example of the use of OSADoEvent, see the figure in section [Using a Script Context to Handle an OSA Event](#).

OSADoEvent - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Example Code](#)
[Glossary](#)

OSADoScript

OSADoScript - Syntax

This method compiles and executes a script and convert the resulting script value to text in a single step rather than calling [OSACompile](#), [OSAExecute](#), and [OSADisplay](#).

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>
```

```

AEDesc      *psourceData;
OSAID       contextID;
DescType    desiredType;
ODSLong     modeFlags;
AEDesc      *presultingText;
OSAError    rc;          /* Return code. */

rc = OSADoScript(psourceData, contextID, desiredType,
                 modeFlags, presultingText);

```

OSADoScript Parameter - psourceData

psourceData ([AEDesc](#) *) - input
 A descriptor record identifying suitable source data for the specified scripting component.

OSADoScript Parameter - contextID

contextID ([OSAID](#)) - input
 The script ID for the context to be used during script execution or kOSANullScript if the scripting component should use its default context.

OSADoScript Parameter - desiredType

desiredType ([DescType](#)) - input
 The desired text descriptor type, such as typeChar, for the resulting descriptor record.

OSADoScript Parameter - modeFlags

modeFlags ([ODSLong](#)) - input
 Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull
 No mode flags are set.

kOSAModeNeverInteract
 Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract
 Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract
 Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer
 Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDisplayForHumans

Resulting text is readable by humans only and cannot be recompiled by OSACompile.

OSADoScript Parameter - presultingText

presultingText ([AEDesc *](#)) - output

The resulting descriptor record.

OSADoScript Return Value - rc

rc ([OSAEError](#)) - returns

Return code.

noErr

No error.

errOSACantCoerce

Data could not be coerced to the requested data type.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

The source data is invalid (syntax error), or an execution error occurred.

OSADoScript - Parameters

psourceData ([AEDesc *](#)) - input

A descriptor record identifying suitable source data for the specified scripting component.

contextID ([OSAID](#)) - input

The script ID for the context to be used during script execution or kOSANullScript if the scripting component should use its default context.

desiredType ([DescType](#)) - input

The desired text descriptor type, such as typeChar, for the resulting descriptor record.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds `kAECanInteract` to `sendMode` parameter of [AESend](#) for events sent when script is executed.

`kOSAModeAlwaysInteract`

Adds `kAEAlwaysInteract` to `sendMode` parameter of [AESend](#) for events sent when script is executed.

`kOSAModeCantSwitchLayer`

Prevents use of `kAECanSwitchLayer` in `sendMode` parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

`kOSAModeDontReconnect`

Adds `kAEDontReconnect` to `sendMode` parameter of [AESend](#) for events sent when script is executed.

`kOSAModeDoRecord`

Prevents use of `kAEDontRecord` in `sendMode` parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

`kOSAModeDisplayForHumans`

Resulting text is readable by humans only and cannot be recompiled by `OSACompile`.

resultingText ([AEDesc *](#)) - output

The resulting descriptor record.

rc ([OSAEError](#)) - returns

Return code.

`noErr`

No error.

`errOSACantCoerce`

Data could not be coerced to the requested data type.

`errOSASystemError`

A general scripting system error occurred.

`errOSAInvalidID`

The specified script ID is invalid.

`errOSAScriptError`

The source data is invalid (syntax error), or an execution error occurred.

OSADoScript - Remarks

Calling the `OSADoScript` method is equivalent to calling [OSACompile](#) followed by [OSAExecute](#) and [OSADisplay](#). After compiling the source data, executing the compiled script using the script context identified by the `contextID` parameter, and returning the text equivalent of the resulting script value in the `resultingText` parameter, `OSADoScript` disposes of both the compiled script and the resulting script value.

If the result code returned by `OSADoScript` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution and the *resultingText* parameter contains the error message associated with the error. In this case, you can obtain more detailed error information by calling [OSAScriptError](#).

Object REXX Notes

The *psourceData* parameter must be of type `typeChar`, the *contextID* parameter must be of `kOSANullScript`, the *desiredType* parameter must be of type `typeChar`, and the *modeFlags* parameter must be `kOSANullScript`.

OSADoScript - Override Policy

A scripting component which sets `kOSASupportsConvenience` in the *componentFlags* field of the component description record must override this method.

OSADoScript - Topics

Class:

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSAExecute

OSAExecute - Syntax

This method executes a compiled script or a script context.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAIID    compiledScriptID;
OSAIID    contextID;
ODSLong   modeFlags;
OSAIID    *presultingScriptValueID;
OSAError   rc;          /* Return code. */

rc = OSAExecute(compiledScriptID, contextID,
                modeFlags, presultingScriptValueID);
```

OSAExecute Parameter - compiledScriptID

compiledScriptID ([OSAIID](#)) - input
 The script ID for the compiled script to be executed.

OSAExecute Parameter - contextID

contextID ([OSAIID](#)) - input
 The script ID for the context to be used during script execution, or kOSANullScript if the scripting component should use its default context.

OSAExecute Parameter - modeFlags

modeFlags (ODSLong) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSAExecute Parameter - presulingScriptValueID

presulingScriptValueID (OSAID *) - output

The script ID for the script value returned or kOSANullScript if the scripting component did not result in a value.

OSAExecute Return Value - rc

rc (OSAError) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

An error occurred during execution.

OSAExecute - Parameters

compiledScriptID (OSAID) - input

The script ID for the compiled script to be executed.

contextID (OSAID) - input

The script ID for the context to be used during script execution, or kOSANullScript if the scripting component should use its default context.

modeFlags (ODSLong) - input

Information used by individual scripting components. This parameter can be set to one of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

resultingScriptValueID (OSAID *) - output

The script ID for the script value returned or kOSANullScript if the scripting component did not result in a value.

rc (OSAError) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

An error occurred during execution.

OSAExecute - Remarks

This method executes the compiled script identified by the *compiledScriptID* parameter, using the script context identified by the *contextID* parameter to maintain state information, such as the binding of variables, for the compiled script. After successfully executing a script, OSAExecute returns the script ID for a resulting script value or, if execution does not result in a value, the constant kOSANullScript.

You can use the [OSACoerceToDesc](#) method to coerce the resulting script value to a descriptor record of a desired descriptor type or the [OSADisplay](#) method to obtain the equivalent source data for the script value.

If the result code returned by OSAExecute is a general result code, there was a problem in arranging for the script to be run. If the result code is errOSAScriptError, an error occurred during script execution. In this case, you can obtain more detailed error information by calling [OSAScriptError](#).

Object REXX Notes

The *contextID* parameter must be kOSANullScript.

OSAExecute - Override Policy

A scripting component must override this method.

OSAExecute - Example Code

For examples of the use of the OSAExecute method, see the first figure in sections [Compiling and Executing Source Data](#) and in section. [Loading and Executing Script Data](#).

OSAExecute - Topics

Class:
 OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Example Code](#)
[Glossary](#)

OSAExecuteEvent

OSAExecuteEvent - Syntax

This method handles an OSA event with the aid of a script context and obtains a script ID for the resulting script value.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAEvent      *ptheOSAEvent;
OSAID         contextID;
ODSLong       modeFlags;
OSAID         *presultingScriptValueID;
OSAError      rc;                /* Return code. */

rc = OSAExecuteEvent(ptheOSAEvent, contextID,
                    modeFlags, presultingScriptValueID);
```

OSAExecuteEvent Parameter - ptheOSAEvent

ptheOSAEvent ([OSAEvent](#) *) - input
 The OSA event to be handled.

OSAExecuteEvent Parameter - contextID

contextID ([OSAID](#)) - input

The script ID of the script context to be used to handle the OSA event.

OSAExecuteEvent Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to any of the following values:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds **kAENeverInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds **kAEAlwaysInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds **kAECanInteract** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of **kAECanSwitchLayer** in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDispatchToDirectObject

Causes the event to be dispatched to the direct object of the event. The direct object (or subject attribute if the direct object is a non-object specifier) is resolved, and the resulting script object is the recipient of the message. The context argument to this method serves as the root of the lookup/resolution process.

It also causes the context argument to be interpreted as the root context in which the direct parameter is resolved in order to find the real context to receive the event (otherwise, the context is used directly to look up handlers).

kOSAModeDontGetDataForArguments

Indicates that components do not have to get data for object specifier arguments. This applies to object specifiers other than those in the direct parameter.

The default behavior for object specifier parameters is to get the data associated with the parameter; however, if this mode flag is set, any object specifiers that are parameters in the events are treated as "references" in the script. The scripting component should not resolve the reference by sending a Get Data event to get the contexts of the reference.

kOSAModeDontReconnect

Adds **kAEDontReconnect** to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of **kAEDontRecord** in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSAExecuteEvent Parameter - presulatingScriptValueID

presulatingScriptValueID ([OSAID](#) *) - output

A script ID for the resulting script value.

OSAExecuteEvent Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr

No error.

errAEventNotHandled

The script context does not contain handler for event.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

An error occurred during execution, or an attempt was made to pass event to a NULL resume dispatch function.

OSAExecuteEvent - Parameters

ptheOSAEvent ([OSAEvent *](#)) - input
The OSA event to be handled.

contextID ([OSAID](#)) - input
The script ID of the script context to be used to handle the OSA event.

modeFlags ([ODSLong](#)) - input
Information used by individual scripting components. This parameter can be set to any of the following values:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDispatchToDirectObject

Causes the event to be dispatched to the direct object of the event. The direct object (or subject attribute if the direct object is a non-object specifier) is resolved, and the resulting script object is the recipient of the message. The context argument to this method serves as the root of the lookup/resolution process.

It also causes the context argument to be interpreted as the root context in which the direct parameter is resolved in order to find the real context to receive the event (otherwise, the context is used directly to look up handlers).

kOSAModeDontGetDataForArguments

Indicates that components do not have to get data for object specifier arguments. This applies to object specifiers other than those in the direct parameter.

The default behavior for object specifier parameters is to get the data associated with the parameter; however, if this mode flag is set, any object specifiers that are parameters in the events are treated as "references" in the script. The scripting component should not resolve the reference by sending a Get Data event to get the contexts of the reference.

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

resultingScriptValueID ([OSAID *](#)) - output

A script ID for the resulting script value.

rc ([OSAEError](#)) - returns

Return code.

noErr

No error.

errAEEEventNotHandled

The script context does not contain handler for event.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSAScriptError

An error occurred during execution, or an attempt was made to pass event to a NULL resume dispatch function.

OSAExecuteEvent - Remarks

This method attempts to use the script context specified by the *contextID* parameter to handle the OSA event specified by the *ptheOSAEvent* parameter. If the scripting component determines that the script context cannot handle the event, this method immediately returns `errAEEEventNotHandled` rather than `errOSAScriptError`.

If the scripting component determines that the script context can handle the event, `OSAExecuteEvent` executes the script context's handler and returns the resulting script ID. If execution of the script context's handler for the event generates an error, `OSAExecuteEvent` returns `errOSAScriptError`, and you can get more detailed error information by calling the [OSAScriptError](#) method.

If the script context identified by the *contextID* parameter specifies that the OSA event should be passed to the application's default handler for that event, `OSAExecuteEvent` passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event—that is, without calling `OSAExecuteEvent` again. If the *contextID* parameter is `kOSANullScript`, the `OSAExecuteEvent` method passes the event directly to the resume dispatch function. If a call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Note: The `OSAExecuteEvent` method can generate the result code `errAEEEventNotHandled` in at least two ways. If the scripting component determines that a script context does not declare a handler for a particular event, `OSAExecuteEvent` immediately returns `errAEEEventNotHandled`. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, `OSAExecuteEvent` returns `errOSAScriptError` and a call to [OSAScriptError](#) with `kOSAErrorNumber` in the selector parameter returns `errAEEEventNotHandled` as the resulting error description.

OSAExecuteEvent - Override Policy

A scripting component which sets `kOSASupportsEventHandling` in the *componentFlags* field of the component description record must override this method.

OSAExecuteEvent - Topics

Class:

OSAScriptingComponent

Select an item:

OSAGetActiveProc

OSAGetActiveProc - Syntax

This method returns a pointer to the active function that a scripting component is currently using.

```
#define INCL_OSASCRIPINGCOMPONENT
#define INCL_OSAAPI
#include <os2.h>

OSActiveUPP      *pactiveProc;
ODSLong          *prefCon;
OSAError         rc;          /* Return code. */

rc = OSAGetActiveProc(pactiveProc, prefCon);
```

OSAGetActiveProc Parameter - pactiveProc

pactiveProc ([OSActiveUPP *](#)) - output

A pointer to the active function currently set for the specified scripting component.

OSAGetActiveProc Parameter - prefCon

prefCon ([ODSLong *](#)) - output

The reference constant associated with the active function for the specified scripting component.

OSAGetActiveProc Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr

No error.

errOSASystemError
A general scripting system error occurred.

OSAGetActiveProc - Parameters

pactiveProc ([OSAActiveUPP *](#)) - output
A pointer to the active function currently set for the specified scripting component.

prefCon ([ODSLong *](#)) - output
The reference constant associated with the active function for the specified scripting component.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.

OSAGetActiveProc - Remarks

Because of the availability of threads, OS/2 does not require that scripting components support this method. It is not supported by Object REXX.

OSAGetActiveProc - Override Policy

A scripting component must override this method.

OSAGetActiveProc - Topics

Class:
OSAScriptingComponent

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSAGetCreateProc

OSAGetCreateProc - Syntax

This method returns a pointer to the create function that a scripting component is currently using to create OSA events.

```
#define INCL_OSASCRIPINGCOMPONENT
#define INCL_OSAAPI
#include <os2.h>

AECreatOSAEventUPP      *pcreateProc;
ODSLong                 *prefCon;
OSAError                 rc;          /* Return code. */

rc = OSAGetCreateProc(pcreateProc, prefCon);
```

OSAGetCreateProc Parameter - pcreateProc

pcreateProc (AECreatOSAEventUPP *) - output
A pointer to the create function currently set for the specified scripting component.

OSAGetCreateProc Parameter - prefCon

prefCon (ODSLong *) - output
The reference constant associated with the create function for the specified scripting component.

OSAGetCreateProc Return Value - rc

rc (OSAError) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.

OSAGetCreateProc - Parameters

pcreateProc (AECreatOSAEventUPP *) - output
A pointer to the create function currently set for the specified scripting component.

prefCon (ODSLong *) - output
The reference constant associated with the create function for the specified scripting component.

rc ([OSAError](#)) - returns
Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSAGetCreateProc - Override Policy

A scripting component which sets kOSASupportsAESending in the *componentFlags* field of the component description record must override this method.

OSAGetCreateProc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Override Policy](#)

[Glossary](#)

OSAGetCurrentDialect

OSAGetCurrentDialect - Syntax

This method returns the dialect code for the dialect currently being used by a scripting component.

```
#define INCL_OSASCRIPINGCOMPONENT
#define INCL_OSAAPI
#include <os2.h>

ODSShort      *presultingDialectCode;
OSAError      rc;                                /* Return code. */

rc = OSAGetCurrentDialect(presultingDialectCode);
```

OSAGetCurrentDialect Parameter - presultingDialectCode

presultingDialectCode ([ODSShort *](#)) - output

The code for the current dialect of the specified scripting component.

OSAGetCurrentDialect Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSANoSuchDialect	Invalid dialect code.

OSAGetCurrentDialect - Parameters

resultingDialectCode ([ODSShort *](#)) - output
The code for the current dialect of the specified scripting component.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSANoSuchDialect	Invalid dialect code.

OSAGetCurrentDialect - Override Policy

A scripting component which sets `kOSASupportsGetSource` in the *componentFlags* field of the component description record must override this method.

OSAGetCurrentDialect - Topics

Class:
[OSAScriptingComponent](#)

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Override Policy](#)
[Glossary](#)

OSAGetDialectInfo

OSAGetDialectInfo - Syntax

This method returns information about a specified dialect provided by the scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ODSShort    dialectCode;
OSType      selector;
AEDesc      *presultingDialectInfo;
OSAError    rc;          /* Return code. */

rc = OSAGetDialectInfo(dialectCode, selector,
    presultingDialectInfo);
```

OSAGetDialectInfo Parameter - dialectCode

dialectCode ([ODSShort](#)) - input

A code for the dialect about which you want information. You can obtain a list of the scripting component's dialect codes by calling [OSAAvailableDialectCodeList](#).

OSAGetDialectInfo Parameter - selector

selector ([OSType](#)) - input

A flag indicating the type of information to be returned in the *presultingDialectInfo* parameter. This constant determines the descriptor type for the descriptor record returned.

All scripting components support the following constants for this parameter:

keyOSADialectName

Used with descriptor record of any text type, such as type typeChar.

keyOSADialectLangCode

Used with descriptor record of type typeShortInteger.

keyOSADialectScriptCode

Used with descriptor record of type typeShortInteger

OSAGetDialectInfo Parameter - presultingDialectInfo

presultingDialectInfo ([AEDesc *](#)) - output

A descriptor record containing the requested information. The descriptor record's descriptor type corresponds to the constant specified in the selector parameter.

OSAGetDialectInfo Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSABadSelector	Invalid selector.
errOSANoSuchDialect	Invalid dialect code.

OSAGetDialectInfo - Parameters

dialectCode ([ODSShort](#)) - input
A code for the dialect about which you want information. You can obtain a list of the scripting component's dialect codes by calling [OSAAvailableDialectCodeList](#).

selector ([OSType](#)) - input
A flag indicating the type of information to be returned in the *resultingDialectInfo* parameter. This constant determines the descriptor type for the descriptor record returned.

All scripting components support the following constants for this parameter:

keyOSADialectName	Used with descriptor record of any text type, such as type typeChar.
keyOSADialectLangCode	Used with descriptor record of type typeShortInteger.
keyOSADialectScriptCode	Used with descriptor record of type typeShortInteger

resultingDialectInfo ([AEDesc *](#)) - output
A descriptor record containing the requested information. The descriptor record's descriptor type corresponds to the constant specified in the selector parameter.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSABadSelector	Invalid selector.
errOSANoSuchDialect	Invalid dialect code.

OSAGetDialectInfo - Remarks

After you obtain a list of dialect codes by calling [OSAAvailableDialectCodeList](#), you can pass any of those codes to OSAGetDialectInfo to get information about the corresponding dialect. The descriptor type of the descriptor record returned by OSAGetDialectInfo depends on the constant specified in the selector parameter.

Individual scripting components might allow you to specify additional constants.

OSAGetDialectInfo - Override Policy

A scripting component which sets kOSASupportsGetSource in the *componentFlags* field of the component description record must override this method.

OSAGetDialectInfo - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSAGetResumeDispatchProc

OSAGetResumeDispatchProc - Syntax

This method returns the resume dispatch function currently being used by a scripting component instance during execution of a scripting component's continue statement or its equivalent.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEEEventHandlerUPP    *presumeDispatchProc;
ODSLong               *prefCon;
OSAError              rc;                                /* Return code. */

rc = OSAGetResumeDispatchProc(presumeDispatchProc,
                              prefCon);
```

OSAGetResumeDispatchProc Parameter - presumeDispatchProc

presumeDispatchProc ([AEEEventHandlerUPP *](#)) - output

A pointer to the resume dispatch function for the specified scripting component. This parameter can return the following value:

kOSAUseStandardDispatch
No resume dispatch function has been registered.

OSAGetResumeDispatchProc Parameter - prefCon

prefCon ([ODSLong *](#)) - output

The reference constant associated with the resume dispatch function.

OSAGetResumeDispatchProc Return Value - rc

rc ([OSAEError](#)) - returns

Return code.

noErr
No error.
errOSASystemError
General scripting system error.

OSAGetResumeDispatchProc - Parameters

presumeDispatchProc ([AEEEventHandlerUPP *](#)) - output

A pointer to the resume dispatch function for the specified scripting component. This parameter can return the following value:

kOSAUseStandardDispatch
No resume dispatch function has been registered.

prefCon ([ODSLong *](#)) - output

The reference constant associated with the resume dispatch function.

rc ([OSAEError](#)) - returns

Return code.

noErr
No error.
errOSASystemError
General scripting system error.

OSAGetResumeDispatchProc - Override Policy

A scripting component which sets **kOSASupportsEventHandling** in the *componentFlags* field of the component description record must override this method.

OSAGetResumeDispatchProc - Topics

Class:
OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Override Policy](#)
[Glossary](#)

OSAGetScriptInfo

OSAGetScriptInfo - Syntax

This method obtains information about script data according to the value you pass in the *selector* parameter.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptID;
OSType     selector;
ODSLong    *presult;
OSAError   rc;      /* Return code. */

rc = OSAGetScriptInfo(scriptID, selector,
                      presult);
```

OSAGetScriptInfo Parameter - scriptID

scriptID ([OSAID](#)) - input
The script ID of the script data whose information is to be returned.

OSAGetScriptInfo Parameter - selector

selector ([OSType](#)) - input
A flag indicating the type of information to be returned. This parameter can be set to any of the following values:

<code>kOSScriptIsModified</code>	A long integer indicating the number of times the script data has been modified since it was passed to OSALoad .
<code>kOSScriptIsTypeCompiledScript</code>	A boolean value indicating whether the script data is a compiled script.
<code>kOSScriptIsTypeScriptValue</code>	A boolean value indicating whether the script data is a script value.
<code>kOSScriptIsTypeScriptContext</code>	A boolean value indicating whether the script data is a script context.
<code>kOSScriptBestType</code>	A descriptor type that can be passed to OSACoerceToDesc <code>formt=extonly</code> .
<code>OSACanGetSource</code>	A boolean value indicating whether the script data can be successfully passed to OSAGetSource .
<code>kASHasOpenHandler</code>	A boolean value indicating whether a script context with the specified script ID contains a handler for the Open Documents event. If the script ID does not identify a script context, <code>OSAGetScriptInfo</code> returns the result code <code>errOSAIllegalAccess</code> .

OSAGetScriptInfo Parameter - result

result ([ODSLong](#) *) - output
The requested information, which can be coerced to the appropriate descriptor type for the value specified in the selector parameter.

OSAGetScriptInfo Return Value - rc

rc ([OSAError](#)) - returns
Return code.

<code>noErr</code>	No error.
<code>errOSASystemError</code>	A general scripting system error occurred.
<code>errOSAInvalidID</code>	The specified script ID is invalid.
<code>errOSABadSelector</code>	The specified selector value is not supported by scripting component.

OSAGetScriptInfo - Parameters

scriptID ([OSAID](#)) - input
The script ID of the script data whose information is to be returned.

selector ([OSType](#)) - input
A flag indicating the type of information to be returned. This parameter can be set to any of the following values:

<code>kOSScriptIsModified</code>	A long integer indicating the number of times the script data has been modified since it was passed to OSALoad .
<code>kOSScriptIsTypeCompiledScript</code>	A boolean value indicating whether the script data is a compiled script.

kOSAScriptIsTypeScriptValue

A boolean value indicating whether the script data is a script value.

kOSAScriptIsTypeScriptContext

A boolean value indicating whether the script data is a script context.

kOSAScriptBestType

A descriptor type that can be passed to [OSACoerceToDesc fmt=extonly](#).

kOSACanGetSource

A boolean value indicating whether the script data can be successfully passed to [OSAGetSource](#).

kASHasOpenHandler

A boolean value indicating whether a script context with the specified script ID contains a handler for the Open Documents event. If the script ID does not identify a script context, OSAGetScriptInfo returns the result code `errOSAllLegalAccess`.

result ([ODSLong](#) *) - output

The requested information, which can be coerced to the appropriate descriptor type for the value specified in the selector parameter.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSABadSelector

The specified selector value is not supported by scripting component.

OSAGetScriptInfo - Remarks

Object REXX Notes

Although you can specify `kOSAScriptIsModified` in the *selector* parameter when you are using the Object REXX component without generating an error, the current version of Object REXX interprets this request conservatively. The Object REXX component stores script data in a network of interlocking structures, and running a script can cause any of these structures to be modified. If you pass a script ID to `OSAGetScriptInfo` with `kOSAScriptIsModified` as the value of the *selector* parameter, the Object REXX component returns 1 if there is any possibility that the script data or related structures may have been modified and 0 if there is no possibility that they have been modified.

OSAGetScriptInfo - Override Policy

Scripting components must override this method.

OSAGetScriptInfo - Topics

Class:

`OSAScriptingComponent`

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSAGetSendProc

OSAGetSendProc - Syntax

This method returns a pointer to the send function that a scripting component is currently using.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSASendUPP    *psendProc;
ODSLong       *prefCon;
OSAError      rc;          /* Return code. */

rc = OSAGetSendProc(psendProc, prefCon);
```

OSAGetSendProc Parameter - psendProc

psendProc (*OSASendUPP **) - output

A pointer to the send function currently set for the specified scripting component.

OSAGetSendProc Parameter - prefCon

prefCon (*ODSLong **) - output

The reference constant associated with the send function for the specified scripting component.

OSAGetSendProc Return Value - rc

rc (*OSAError*) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSAGetSendProc - Parameters

psendProc ([OSASendUPP *](#)) - output

A pointer to the send function currently set for the specified scripting component.

prefCon ([ODSLong *](#)) - output

The reference constant associated with the send function for the specified scripting component.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

OSAGetSendProc - Override Policy

A scripting component which sets `kOSASupportsAESending` in the *componentFlags* field of the component description record must override this method.

OSAGetSendProc - Topics

Class:

`OSAScriptingComponent`

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Override Policy](#)

[Glossary](#)

OSAGetSource

OSAGetSource - Syntax

This method decompiles the script data identified by the specified script ID and obtains the equivalent source data.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>
```

```
OSAID      scriptID;
DescType   desiredType;
AEDesc     *presultingSourceData;
OSAError   rc;          /* Return code. */

rc = OSAGetSource(scriptID, desiredType, presultingSourceData);
```

OSAGetSource Parameter - scriptID

scriptID ([OSAID](#)) - input

The script ID for the script data to be decompiled or kOSANullScript if a null source description (such as an empty text string) is returned.

OSAGetSource Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type of the resulting descriptor record. The value typeBest should be used if any type will do.

OSAGetSource Parameter - presultingSourceData

presultingSourceData ([AEDesc *](#)) - output

The resulting descriptor record.

OSAGetSource Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSASourceNotAvailable

The source data not available.

OSAGetSource - Parameters

scriptId ([OSAID](#)) - input

The script ID for the script data to be decompiled or kOSANullScript if a null source description (such as an empty text string) is returned.

desiredType ([DescType](#)) - input

The desired descriptor type of the resulting descriptor record. The value typeBest should be used if any type will do.

resultingSourceData ([AEDesc *](#)) - output

The resulting descriptor record.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

A general scripting system error occurred.

errOSAInvalidID

The specified script ID is invalid.

errOSASourceNotAvailable

The source data not available.

OSAGetSource - Remarks

The OSAGetSource method decompiles the script data identified by the specified script ID and returns a descriptor record containing the equivalent source data. The source data returned need not be exactly the same as the source data originally passed to [OSACompile](#) (for example, white space and formatting might be different), but it should be a reasonable equivalent suitable for user viewing and editing.

The difference between [OSACoerceToDesc](#) and OSAGetSource is that OSAGetSource creates source data that can be displayed to a user or compiled and executed to generate an appropriate value, whereas [OSACoerceToDesc](#) actually returns the value. For example, if you call OSAGetSource and specify a string value, it returns the text surrounded by quotation marks (so that it can be properly compiled). If you call [OSACoerceToDesc](#) and specify a string value, it simply returns the text.

The main difference between [OSADisplay](#) and OSAGetSource is that OSAGetSource can coerce any form of script data using a variety of descriptor types, whereas [OSADisplay](#) can coerce only script values and always produces a descriptor record of a text descriptor type.

Object REXX Notes

The *desiredType* parameter must be of type typeChar.

OSAGetSource - Override Policy

A scripting component which sets kOSASupportsGetSource in the *componentFlags* field of the component description record must override this method.

OSAGetSource - Related Methods

Related Methods

- [OSAScriptingComponent::OSACoerceToDesc](#)
 - [OSAScriptingComponent::OSADisplay](#)
-

OSAGetSource - Example Code

For an example of the use of OSAGetSource, see the figure in section [Modifying and Recompiling a Compiled Script](#).

OSAGetSource - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Example Code](#)

[Related Methods](#)

[Glossary](#)

OSAGetStorageType

OSAGetStorageType - Syntax

This method returns the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

ODUByte      *scriptData;
ODULong      size;
DescType     *pdtype;
OSErr        rc;          /* Return code. */

rc = OSAGetStorageType(scriptData, size, pdtype);
```

OSAGetStorageType Parameter - scriptData

scriptData ([ODUByte](#) *) - input
A pointer to the script data.

OSAGetStorageType Parameter - size

size ([ODULong](#)) - input
The size of the script data.

OSAGetStorageType Parameter - pdtype

pdtype ([DescType *](#)) - output
The descriptor type specified in the script data trailer.

OSAGetStorageType Return Value - rc

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSABadStorageType	Bad storage type.

OSAGetStorageType - Parameters

scriptData ([ODUByte *](#)) - input
A pointer to the script data.

size ([ODULong](#)) - input
The size of the script data.

pdtype ([DescType *](#)) - output
The descriptor type specified in the script data trailer.

rc ([OSErr](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSABadStorageType	Bad storage type.

OSAGetStorageType - Remarks

This method retrieves the scripting component subtype from the trailer. If no trailer can be found, OSAGetStorageType returns the error errOSABadStorageType.

A scripting component does not have to instantiate an instance of the [GenericScriptingComponent](#) to use this method.

OSAGetStorageType - Override Policy

A derived class should not override this method.

OSAGetStorageType - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSALoad

OSALoad - Syntax

This method loads script data.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *pscriptData;
ODSLong     modeFlags;
OSAID       *presultingScriptID;
OSAError    rc;          /* Return code. */

rc = OSALoad(pscriptData, modeFlags, presultingScriptID);
```

OSALoad Parameter - pscriptData

pscriptData ([AEDesc](#) *) - in/out

The descriptor record containing the script data to be loaded.

OSALoad Parameter - modeFlags

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to any of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModePreventGetSource

Only the minimum script data required to run the script should be loaded. This flag is used when either space efficiency is desired, or a script is to be locked so that its implementation may not be viewed.

OSALoad Parameter - presultingScriptID

presultingScriptID ([OSAID *](#)) - output

The script ID of the compiled script.

OSALoad Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSACorruptData

Corrupt data.

errOSASystemError

General scripting system error.

errOSABadStorageType

Script data not for this scripting component.

errOSADataFormatObsolete

Data format is obsolete.

errOSADataFormatTooNew

Data format is too new.

OSALoad - Parameters

pscriptData ([AEDesc *](#)) - in/out

The descriptor record containing the script data to be loaded.

modeFlags ([ODSLong](#)) - input

Information used by individual scripting components. This parameter can be set to any of the following mode flags:

kOSAModeNull

No mode flags are set.

kOSAModePreventGetSource

Only the minimum script data required to run the script should be loaded. This flag is used when either space

efficiency is desired, or a script is to be locked so that its implementation may not be viewed.

presultingScriptID ([OSAID *](#)) - output
The script ID of the compiled script.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACorruptData	Corrupt data.
errOSASystemError	General scripting system error.
errOSABadStorageType	Script data not for this scripting component.
errOSADataFormatObsolete	Data format is obsolete.
errOSADataFormatTooNew	Data format is too new.

OSALoad - Remarks

This method loads script data and returns a script ID. The generic scripting component uses the descriptor record in the `scriptData` parameter to determine which scripting component should load the script. If the descriptor record is of type `typeOSAGenericStorage`, the generic scripting component uses the trailer at the end of the script data to identify the scripting component. If the descriptor record's type is the subtype value for another scripting component, the generic scripting component uses the descriptor type to identify the scripting component.

If you want the script ID returned by `OSALoad` to identify only the minimum script data required to run the script and you are sure that you will not need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the `modeFlags` parameter.

Scripting components other than the generic scripting component can load script data only if it has been saved in a descriptor record whose descriptor type matches the scripting component's subtype.

Script data may change after it has been loaded (for example, your application might allow the user to edit a script's source data). To test whether script data has been modified, pass its script ID to [OSAGetScriptInfo](#). If it has changed, you can call [OSAStore](#) again to obtain a handle to the modified script data and then save it.

Object REXX Notes

The *scriptData* parameter must be of type `typeOREXStorage` or `typeOSAGenericStorage`.

OSALoad - Override Policy

A scripting component must override this method.

OSALoad - Topics

Class:
[OSAScriptingComponent](#)

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSALoadExecute

OSALoadExecute - Syntax

This method loads and executes a script.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *pscriptData;
OSAID       contextID;
ODSLong     modeFlags;
OSAID       *presultingScriptValueID;
OSAError    rc; /* Return code. */

rc = OSALoadExecute(pscriptData, contextID,
                    modeFlags, presultingScriptValueID);
```

OSALoadExecute Parameter - pscriptData

pscriptData ([AEDesc *](#)) - input
The descriptor record identifying the script data to be loaded and executed.

OSALoadExecute Parameter - contextID

contextID ([OSAID](#)) - input
The script ID for the context to be used during script execution or kOSANullScript if the scripting component should use its default context.

OSALoadExecute Parameter - modeFlags

modeFlags ([ODSLong](#)) - input
Information used by individual scripting components. This parameter can be set to any of the following mode flags to control the way in which the scripting component executes:

kOSAModeNull
No mode flags are set.

kOSAModeNeverInteract
Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract
Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract
Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer
Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect
Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord
Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

OSALoadExecute Parameter - presulatingScriptValueID

presulatingScriptValueID ([OSAID](#) *) - output
The script ID for the script value returned.

OSALoadExecute Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr
No error.

errOSACorruptData
Same as errAECorruptData.

errOSASystemError
General scripting system error.

errOSAInvalidID
Invalid script ID.

errOSABadStorageType
Script data not for this scripting component.

errOSAScriptError
Error occurred during execution.

errOSADataFormatObsolete
Data format is obsolete.

errOSADataFormatTooNew
Data format is too new.

OSALoadExecute - Parameters

pscriptData ([AEDesc](#) *) - input
The descriptor record identifying the script data to be loaded and executed.

contextID ([OSAID](#)) - input
The script ID for the context to be used during script execution or kOSANullScript if the scripting component should use its default

context.

modeFlags (ODSLong) - input

Information used by individual scripting components. This parameter can be set to any of the following mode flags to control the way in which the scripting component executes:

kOSAModeNull

No mode flags are set.

kOSAModeNeverInteract

Adds kAENeverInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCanInteract

Adds kAECanInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeAlwaysInteract

Adds kAEAlwaysInteract to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeCantSwitchLayer

Prevents use of kAECanSwitchLayer in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

kOSAModeDontReconnect

Adds kAEDontReconnect to *sendMode* parameter of [AESend](#) for events sent when script is executed.

kOSAModeDoRecord

Prevents use of kAEDontRecord in *sendMode* parameter of [AESend](#) for events sent when script is executed (the opposite of the OSA Event Manager's interpretation of the same bit).

resultingScriptValueID (OSAID *) - output

The script ID for the script value returned.

rc (OSAError) - returns

Return code.

noErr

No error.

errOSACorruptData

Same as errAECorruptData.

errOSASystemError

General scripting system error.

errOSAInvalidID

Invalid script ID.

errOSABadStorageType

Script data not for this scripting component.

errOSAScriptError

Error occurred during execution.

errOSADataFormatObsolete

Data format is obsolete.

errOSADataFormatTooNew

Data format is too new.

OSALoadExecute - Remarks

This method loads script data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, this method disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant kOSANullScript.

If the result code returned by this method is a general result code, there was a problem in arranging for the script to be run. If the result code is [errOSAScriptError](#), an error occurred during script execution. In this case, you can obtain more detailed error information by calling [OSAScriptError](#).

Object REXX Notes

The *pscriptData* parameter must be of type typeOREXXStorage or typeOSAGenericStorage, and the *contextID* parameter must be of kOSANullScript.

OSALoadExecute - Override Policy

A scripting component which sets kOSASupportsConvenience in the *componentFlags* field of the component description record must override this method.

OSALoadExecute - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSAMakeContext

OSAMakeContext - Syntax

This method returns a script ID for a new script context.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *pcontextName;
OSAID       parentContext;
OSAID       *presultingContextID;
OSAError    rc;          /* Return code. */

rc = OSAMakeContext(pcontextName, parentContext,
                    presultingContextID);
```

OSAMakeContext Parameter - pcontextName

pcontextName ([AEDesc *](#)) - input

The name of the new context which some scripting components can use for semantic purposes. This parameter can also be set to the following value:

typeNull

An unnamed context is created.

OSAMakeContext Parameter - parentContext

parentContext ([OSAID](#)) - input

The existing context from which new context inherits bindings or kOSANullScript if the new context does not inherit bindings from any other context.

OSAMakeContext Parameter - presulatingContextID

presulatingContextID ([OSAID *](#)) - output

A script ID for the resulting script context.

OSAMakeContext Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSACantCoerce

Invalid context name.

errOSASystemError

General scripting system error.

errOSAInvalidID

Invalid script ID.

OSAMakeContext - Parameters

pcontextName ([AEDesc *](#)) - input

The name of the new context which some scripting components can use for semantic purposes. This parameter can also be set to the following value:

typeNull

An unnamed context is created.

parentContext ([OSAID](#)) - input

The existing context from which new context inherits bindings or kOSANullScript if the new context does not inherit bindings from any other context.

presulatingContextID ([OSAID *](#)) - output

A script ID for the resulting script context.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSACantCoerce
Invalid context name.
errOSASystemError
General scripting system error.
errOSAInvalidID
Invalid script ID.

OSAMakeContext - Remarks

This method creates a new script context that you may pass to [OSAExecute](#) or [OSAExecuteEvent](#). The new script context inherits the bindings of the script context specified in the *parentContext* parameter.

Special Considerations

If you call OSAMakeContext using an instance of the generic scripting component, the generic scripting component uses the default scripting component to create the new script context.

To compile existing source data into a script context, use [OSACompile](#).

OSAMakeContext - Override Policy

A scripting component which sets kOSASupportsEventHandling in the *componentFlags* field of the component description record must override this method.

OSAMakeContext - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSARemoveStorageType

OSARemoveStorageType - Syntax

This method removes a trailer from the script data in a generic storage descriptor record.

```
#define INCL_ODAPI  
#define INCL_OSACOMPONENT
```

```

#define INCL_OSA_API
#define INCL_OSA
#include <os2.h>

ODUByte      *oldScriptData;
ODULong       oldSize;
ODUByte      *newScriptData;
ODULong       *newSize;
OSErr        rc;          /* Return code. */

rc = OSARemoveStorageType(oldScriptData, oldSize,
                          newScriptData, newSize);

```

OSARemoveStorageType Parameter - oldScriptData

oldScriptData (ODUByte *) - input
A pointer to the old script data.

OSARemoveStorageType Parameter - oldSize

oldSize (ODULong) - input
The size of the old script data.

OSARemoveStorageType Parameter - newScriptData

newScriptData (ODUByte *) - output
A pointer to memory allocated by the user to hold the script data plus the storage type. If this parameter is passed as NULL, the required size is returned in the *newSize* parameter.

OSARemoveStorageType Parameter - newSize

newSize (ODULong *) - in/out
The size of the new script data.

OSARemoveStorageType Return Value - rc

rc (OSErr) - returns
Return code.

noErr

	No error.
errOSASystemError	A general scripting system error occurred.
errOSABufferTooSmall	The size of the buffer is not large enough to hold the data to be returned.

OSARemoveStorageType - Parameters

oldScriptData ([ODUByte *](#)) - input
A pointer to the old script data.

oldSize ([ODULong](#)) - input
The size of the old script data.

newScriptData ([ODUByte *](#)) - output
A pointer to memory allocated by the user to hold the script data plus the storage type. If this parameter is passed as NULL, the required size is returned in the *newSize* parameter.

newSize ([ODULong *](#)) - in/out
The size of the new script data.

rc ([OSError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	A general scripting system error occurred.
errOSABufferTooSmall	The size of the buffer is not large enough to hold the data to be returned.

OSARemoveStorageType - Remarks

This method removes an existing trailer (reducing the handle's size). If no trailer can be found, then the handle is not modified, and noErr is returned.

A scripting component does not have to instantiate an instance of the [GenericScriptingComponent](#) to use this method.

OSARemoveStorageType - Override Policy

A derived class should not override this method.

OSARemoveStorageType - Topics

Class:
[OSAScriptingComponent](#)

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)

OSAScriptError

OSAScriptError - Syntax

This method returns information about errors that occur during script compilation and execution.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSType      selector;
DescType    desiredType;
AEDesc      *presultingErrorDescription;
OSAError    rc;                                /* Return code. */

rc = OSAScriptError(selector, desiredType,
                    presultingErrorDescription);
```

OSAScriptError Parameter - selector

selector ([OSType](#)) - input

A value that determines what is to be returned. This parameter can be set to one of the following constants. The second column describes the information returned in the *presultingErrorDescription* parameter.

- | | |
|--------------------------|--|
| kOSAErrorApp | Either the name or the process ID (PID) of the application that received the error, if it was the result of sending an OSA event. The value of <i>desiredType</i> must be typePID (for the PID) or a text descriptor type such as typeChar (for the name). |
| kOSAErrorBriefMessage | Brief error message associated with error number, excluding the name of the application, any partial result, and the offending object. The value of <i>desiredType</i> must be typeChar or another text descriptor type. |
| kOSAErrorExpectedType | Error used to determine the type expected by a coercion operation if a type error occurred. |
| kOSAErrorMessage | Error message associated with error number, including both the name of the application and a description of the error. This constant is sufficient for simple error reporting. The value of <i>desiredType</i> must be typeChar or another text descriptor type. |
| kOSAErrorNumber | Error number for either system error or scripting component error. The value of <i>desiredType</i> must be typeShortInteger. |
| kOSAErrorOffendingObject | An object specifier record for the object that caused the error. The value of <i>desiredType</i> must be typeObjectSpecifier, typeBest, or typeWildcard. For some scripting components, including Object REXX, these three values are equivalent. |

kOSAErrorPartialResult

Partial result returned after a call to [AESend](#) that failed. This consists of a reply parameter that contains some but not all of the information requested. The value of *desiredType* must be typeBest (for the best type) or typeWildcard (for the default type).

kOSAErrorRange

Range of source data in which error occurred. The value of *desiredType* must be typeOSAErrorRange. Every scripting component should support calls to OSAScriptError that pass kOSAErrorNumber, kOSAErrorMessage, or kOSAErrorPartialResult in the *selector* parameter. Some scripting components may also support calls that pass other values in the *selector* parameter, including kOSAErrorRange, which provides start and end positions delimiting the errant expression in the source data.

OSAScriptError Parameter - desiredType

desiredType ([DescType](#)) - input

The desired descriptor type of the resulting descriptor record. The description in the *resultingErrorDescription* parameter explains how this is determined by the value passed in the *selector* parameter.

OSAScriptError Parameter - resultingErrorDescription

resultingErrorDescription ([AEDesc *](#)) - output

The resulting descriptor record. If the value of the *selector* parameter is kOSAErrorNumber, scripting components might return one of the following general error codes. Although scripting components are not required to support these error codes, their use simplifies error handling for applications that run scripts created by multiple components.

errOSACantCoerce

Same as errAECoercionFail; cannot coerce data to requested descriptor type.

errOSACantAccess

Same as errAENoSuchObject; run-time error in resolution of object specifier record.

errOSAGeneralError

General run-time error.

errOSADivideByZero

Attempt to divide by zero.

errOSANumericOverflow

Numeric overflow.

errOSACantLaunch

Cannot launch specified file because it is not an application.

errOSAAppNotHighLevelEventAware

Does not respond to OSA events.

errOSACorruptTerminology

The application has a corrupted **aete** resource.

errOSAStackOverflow

Stack overflow.

errOSAInternalTableOverflow

Internal table overflow.

errASDataBlockTooLarge

Attempt to create a value larger than the allowable size.

errOSATypeError

Same as errAEWrongDataType; wrong descriptor type.

errOSAMessageNotUnderstood	Same as errAEEventNotHandled; event not handled or message not understood.
errOSAUndefinedMessage	Same as errAEHandlerNotFound; handler not found for message.
errOSAlllegalIndex	Same as errAEIllegalIndex; not a valid index.
errOSAlllegalRange	Same as errAEImpossibleRange; range of specified objects not possible..
errOSASyntaxError	General syntax error.
errOSASyntaxTypeError	Syntax error; parser expected one type but found another.
errOSATokenTooLong	Identifier too long.
errOSAMissingParameter	Same as errAEDescNotFound; descriptor record not found.
errOSAParameterMismatch	Same as errAEWrongNumberArgs; wrong number of arguments.
errOSADuplicateParameter	Parameter specified more than once.
errOSADuplicateProperty	Property specified more than once.
errOSADuplicateHandler	Handler defined more than once.
errOSAUndefinedVariable	Undefined variable.
errOSAINconsistentDeclarations	Inconsistent declarations.
errOSAControlFlowError	Control flow error.

If the value of the *selector* parameter is kOSAErrorNumber, the Object REXX component might return one of the following error codes:

errAECantConsiderAndIgnore	Cannot both consider and ignore a parameter.
errASCantCompareMoreThan32k	Cannot compare text larger than 32K.
errASCantCompareMixedScripts	Cannot compare text from different script systems.
errASTerminologyNestingTooDeep	Tell statements nested too deeply.
errASInconsistentNames	Syntax error; names at beginning and end of handler are inconsistent (Object REXX English dialect only).

OSAScriptError Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr

	No error.
errOSACantCoerce	Desired type not supported by scripting component.
errOSASystemError	General scripting system error.
errOSABadSelector	Selector value not supported by scripting component.

OSAScriptError - Parameters

selector (*OSType*) - input

A value that determines what is to be returned. This parameter can be set to one of the following constants. The second column describes the information returned in the *resultingErrorDescription* parameter.

kOSAErrorApp	Either the name or the process ID (PID) of the application that received the error, if it was the result of sending an OSA event. The value of <i>desiredType</i> must be typePID (for the PID) or a text descriptor type such as typeChar (for the name).
kOSAErrorBriefMessage	Brief error message associated with error number, excluding the name of the application, any partial result, and the offending object. The value of <i>desiredType</i> must be typeChar or another text descriptor type.
kOSAErrorExpectedType	Error used to determine the type expected by a coercion operation if a type error occurred.
kOSAErrorMessage	Error message associated with error number, including both the name of the application and a description of the error. This constant is sufficient for simple error reporting. The value of <i>desiredType</i> must be typeChar or another text descriptor type.
kOSAErrorNumber	Error number for either system error or scripting component error. The value of <i>desiredType</i> must be typeShortInteger.
kOSAErrorOffendingObject	An object specifier record for the object that caused the error. The value of <i>desiredType</i> must be typeObjectSpecifier, typeBest, or typeWildcard. For some scripting components, including Object REXX, these three values are equivalent.
kOSAErrorPartialResult	Partial result returned after a call to AESend that failed. This consists of a reply parameter that contains some but not all of the information requested. The value of <i>desiredType</i> must be typeBest (for the best type) or typeWildcard (for the default type).
kOSAErrorRange	Range of source data in which error occurred. The value of <i>desiredType</i> must be typeOSAErrorRange. Every scripting component should support calls to OSAScriptError that pass kOSAErrorNumber, kOSAErrorMessage, or kOSAErrorPartialResult in the <i>selector</i> parameter. Some scripting components may also support calls that pass other values in the <i>selector</i> parameter, including kOSAErrorRange, which provides start and end positions delimiting the errant expression in the source data.

desiredType (*DescType*) - input

The desired descriptor type of the resulting descriptor record. The description in the *resultingErrorDescription* parameter explains how this is determined by the value passed in the *selector* parameter.

resultingErrorDescription (*AEDesc **) - output

The resulting descriptor record. If the value of the *selector* parameter is kOSAErrorNumber, scripting components might return one of the following general error codes. Although scripting components are not required to support these error codes, their use simplifies error handling for applications that run scripts created by multiple components.

errOSACantCoerce	Same as errAEO coercionFail; cannot coerce data to requested descriptor type.
errOSACantAccess	Same as errAENoSuchObject; run-time error in resolution of object specifier record.
errOSAGeneralError	

	General run-time error.
errOSADivideByZero	Attempt to divide by zero.
errOSANumericOverflow	Numeric overflow.
errOSACantLaunch	Cannot launch specified file because it is not an application.
errOSAAppNotHighLevelEventAware	Does not respond to OSA events.
errOSACorruptTerminology	The application has a corrupted aete resource.
errOSAStackOverflow	Stack overflow.
errOSAInternalTableOverflow	Internal table overflow.
errASDataBlockTooLarge	Attempt to create a value larger than the allowable size.
errOSATypeError	Same as errAEWrongDataType; wrong descriptor type.
errOSAMessageNotUnderstood	Same as errAEEEventNotHandled; event not handled or message not understood.
errOSAUndefinedMessage	Same as errAEHandlerNotFound; handler not found for message.
errOSAIllegalIndex	Same as errAEIllegalIndex; not a valid index.
errOSAIllegalRange	Same as errAEImpossibleRange; range of specified objects not possible..
errOSASyntaxError	General syntax error.
errOSASyntaxTypeError	Syntax error; parser expected one type but found another.
errOSATokenTooLong	Identifier too long.
errOSAMissingParameter	Same as errAEDescNotFound; descriptor record not found.
errOSAParameterMismatch	Same as errAEWrongNumberArgs; wrong number of arguments.
errOSADuplicateParameter	Parameter specified more than once.
errOSADuplicateProperty	Property specified more than once.
errOSADuplicateHandler	Handler defined more than once.
errOSAUndefinedVariable	Undefined variable.
errOSAInconsistentDeclarations	Inconsistent declarations.
errOSAControlFlowError	Control flow error.

If the value of the *selector* parameter is kOSAErrorNumber, the Object REXX component might return one of the following error codes:

errAECantConsiderAndIgnore	Cannot both consider and ignore a parameter.
errASCantCompareMoreThan32k	Cannot compare text larger than 32K.
errASCantCompareMixedScripts	Cannot compare text from different script systems.
errASTerminologyNestingTooDeep	Tell statements nested too deeply.
errASInconsistentNames	Syntax error; names at beginning and end of handler are inconsistent (Object REXX English dialect only).

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSACantCoerce	Desired type not supported by scripting component.
errOSASystemError	General scripting system error.
errOSABadSelector	Selector value not supported by scripting component.

OSAScriptError - Remarks

[OSACompile](#) returns errOSAScriptError when there is a syntax error exists in the script data. When a method, such as [OSAExecute](#), returns errOSAScriptError, a run-time error has occurred. OSAScriptError can be called in either case to obtain the detailed error information available from the scripting component.

When a method, such as [OSAExecute](#), returns the error errOSAScriptError, the OSAScriptError method can be used to get more specific information about the error from the scripting component that encountered it. (This information remains available only until the next call to the same scripting component.) The information returned by OSAScriptError depends on the value passed in the selector parameter, which also determines the descriptor type you should specified in the *desiredType* parameter.

If the value of the *selector* parameter is kOSAErrorRange, the value of *desiredType* must be typeOSAErrorRange.

```
#define kOSAErrorRange          0x676E7265      /* 'erng' */
```

A descriptor record of type typeOSAErrorRange is an AE record that consists of two descriptor records of typeShortInteger specified by these keywords:

```
#define keyOSASourceStart      0x73637273      /* 'srcs' */
#define keyOSASourceEnd       0x65637273      /* 'srce' */
```

Special Considerations

If you call this method using an instance of the generic scripting component, the generic scripting component uses the same instance of the scripting component that it used for the previous call.

OSAScriptError can be called for both compile-time errors (for scripting components that support compiling) and run-time errors.

[OSACompile](#) returns errOSAScriptError when there is a syntax error exists in the script data. When a method, such as [OSAExecute](#), returns errOSAScriptError, a run-time error has occurred. OSAScriptError can be called in either case to obtain the detailed error information available from the scripting component.

Scripting components that do not support compilation report both syntax and run-time errors after a call to the OSAScriptError method. The error number and message reply parameters indicate which type of error has occurred.

Developers are encouraged to provide meaningful error message for scripting components by reporting data in optional reply parameters, such as kOSAErrorOffendingObject and kOSAErrorExpectedType, in addition to the error number and error message reply parameters. This

creates more meaningful error display dialogs and messages for the user.

Object REXX Notes

A *selector* of type:

- kOSAErrorNumber returns the Object REXX Minor error code.
- kOSAErrorMessage returns the Object REXX Minor error text.
- kOSAErrorBriefMessage returns the Object REXX Major error text.

All other values produce an error return code (errOSABadSelector).

OSAScriptError - Override Policy

A scripting component must override this method.

OSAScriptError - Example Code

For an example of the use of the OSAScriptError method, see the second figure in section [Compiling and Executing Source Data](#).

OSAScriptError - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Example Code](#)
[Glossary](#)

OSAScriptingComponentName

OSAScriptingComponentName - Syntax

This method returns the name of the scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *presultingScriptingComponentName;
OSAError    rc;                                /* Return code. */
```

```
rc = OSAScriptingComponentName (presultingScriptingComponentName);
```

OSAScriptingComponentName Parameter - presultingScriptingCom

presultingScriptingComponentName ([AEDesc *](#)) - output

The name of the scripting component or, if the component is the generic scripting component, the name of the default scripting component.

OSAScriptingComponentName Return Value - rc

rc ([OSAEError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSAScriptingComponentName - Parameters

presultingScriptingComponentName ([AEDesc *](#)) - output

The name of the scripting component or, if the component is the generic scripting component, the name of the default scripting component.

rc ([OSAEError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSAScriptingComponentName - Remarks

This method returns a descriptor record that can be coerced to a text descriptor type, such as typeChar. This can be useful if you want to display the name of the scripting language in which the user should write a new script.

For an example of the use of OSAScriptingComponentName, see the first figure in section [Compiling and Executing Source Data](#).

OSAScriptingComponentName - Override Policy

A scripting component which sets kOSASupportsCompiling in the *componentFlags* field of the component description record must override

this method.

OSAScriptingComponentName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSASetActiveProc

OSASetActiveProc - Syntax

This method sets the active function that a scripting component calls periodically while executing a script.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAActiveUPP    activeProc;
ODSLong         refCon;
OSAError        rc;          /* Return code. */

rc = OSASetActiveProc(activeProc, refCon);
```

OSASetActiveProc Parameter - activeProc

activeProc ([OSAActiveUPP](#)) - input

A pointer to the active function to set. If the value of this parameter is NULL, the scripting component's default active function is set.

OSASetActiveProc Parameter - refCon

refCon ([ODSLong](#)) - input

A reference constant to be associated with the active function. This parameter can be used for many purposes; for instance, it could contain a handle to data used by the active function.

OSASetActiveProc Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetActiveProc - Parameters

activeProc ([OSAActiveUPP](#)) - input

A pointer to the active function to set. If the value of this parameter is NULL, the scripting component's default active function is set.

refCon ([ODSLong](#)) - input

A reference constant to be associated with the active function. This parameter can be used for many purposes; for instance, it could contain a handle to data used by the active function.

rc ([OSAError](#)) - returns
Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetActiveProc - Remarks

Because of the availability of threads, OS/2 does not require scripting components to support this method. It is not supported by Object REXX.

OSASetActiveProc - Override Policy

A scripting component must override this method.

OSASetActiveProc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

OSASetCreateProc

OSASetCreateProc - Syntax

This method specifies a create function that a scripting component should use instead of the OSA Event Manager's [AECreatOSAEvent](#) function when creating OSA events.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AECreatOSAEventUPP    createProc;
ODSLong               refCon;
OSAError              rc;          /* Return code. */

rc = OSASetCreateProc(createProc, refCon);
```

OSASetCreateProc Parameter - createProc

createProc ([AECreatOSAEventUPP](#)) - input
A pointer to the create function to be used when creating OSA events.

OSASetCreateProc Parameter - refCon

refCon ([ODSLong](#)) - input
A reference constant to be associated with the create function.

OSASetCreateProc Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.

OSASetCreateProc - Parameters

createProc (AECreatEOSAEventUPP) - input
A pointer to the create function to be used when creating OSA events.

refCon (ODSLong) - input
A reference constant to be associated with the create function.

rc (OSAEError) - returns
Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetCreateProc - Override Policy

A scripting component which sets kOSASupportsAESending in the *componentFlags* field of the component description record must override this method.

OSASetCreateProc - Topics

Class:
OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Override Policy](#)

[Glossary](#)

OSASetCurrentDialect

OSASetCurrentDialect - Syntax

This method sets the current dialect for a scripting component.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
```

```

#define INCL_OSA
#include <os2.h>

ODSShort    dialectCode;
OSAError    rc;          /* Return code. */

rc = OSASetCurrentDialect(dialectCode);

```

OSASetCurrentDialect Parameter - dialectCode

dialectCode (ODSShort) - input
The code of the dialect to be set.

OSASetCurrentDialect Return Value - rc

rc (OSAError) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSANoSuchDialect	Invalid dialect code.

OSASetCurrentDialect - Parameters

dialectCode (ODSShort) - input
The code of the dialect to be set.

rc (OSAError) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSANoSuchDialect	Invalid dialect code.

OSASetCurrentDialect - Override Policy

A scripting component which sets `kOSASupportsAESending` in the *componentFlags* field of the component description record must override this method.

OSASetCurrentDialect - Topics

Class:
OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Override Policy](#)
[Glossary](#)

OSASetDefaultTarget

OSASetDefaultTarget - Syntax

This method sets the default target application for OSA events.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

AEAddressDesc      *ptarget;
OSAError           rc;      /* Return code. */

rc = OSASetDefaultTarget(ptarget);
```

OSASetDefaultTarget Parameter - ptarget

ptarget ([AEAddressDesc](#) *) - input

The address of the application that is being made the default application. If you pass a NULL descriptor record in this parameter, the scripting component treats the current process as the default target.

OSASetDefaultTarget Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr
No error.
errOSASystemError
General scripting system error.

OSASetDefaultTarget - Parameters

ptarget ([AEAddressDesc *](#)) - input

The address of the application that is being made the default application. If you pass a NULL descriptor record in this parameter, the scripting component treats the current process as the default target.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetDefaultTarget - Remarks

This method establishes the default target application for OSA event sending and the default application from which the scripting component should obtain terminology information.

If your application does not call this function or if you pass a null-descriptor record in the target parameter, the scripting component treats the current process-that is, the application that calls [OSAExecute](#) or related routines-as the default target application.

OSASetDefaultTarget - Override Policy

A scripting component which sets kOSASupportsAESending in the *componentFlags* field of the component description record must override this method.

OSASetDefaultTarget - Topics

Class:

[OSAScriptingComponent](#)

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSASetScriptInfo

OSASetScriptInfo - Syntax

This method sets specified information about script data.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptID;
OSType     selector;
ODSLong    value;
OSAError    rc;          /* Return code. */

rc = OSASetScriptInfo(scriptID, selector,
                      value);
```

OSASetScriptInfo Parameter - scriptID

scriptID ([OSAID](#)) - input

The script ID of the script data whose information is to be set.

OSASetScriptInfo Parameter - selector

selector ([OSType](#)) - input

A flag indicating the type of information to be set. All scripting components can accept this value:

kOSAScriptsModified

(**modi**) Indicates that the number of changes since the script data was loaded or created should be set to the value in the *value* parameter. The Object REXX component provides limited support for this constant.

OSASetScriptInfo Parameter - value

value ([ODSLong](#)) - input

The value to set.

OSASetScriptInfo Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

errOSAInvalidID Invalid script ID.
errOSABadSelector Selector value not supported by scripting component.

OSASetScriptInfo - Parameters

scriptID ([OSAID](#)) - input

The script ID of the script data whose information is to be set.

selector ([OSType](#)) - input

A flag indicating the type of information to be set. All scripting components can accept this value:

kOSAScriptIsModified

(**modi**) Indicates that the number of changes since the script data was loaded or created should be set to the value in the *value* parameter. The Object REXX component provides limited support for this constant.

value ([ODSLong](#)) - input

The value to set.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

errOSAInvalidID

Invalid script ID.

errOSABadSelector

Selector value not supported by scripting component.

OSASetScriptInfo - Remarks

This method sets script information according to the value passed in the *selector* parameter. If you use the kOSAScriptIsModified constant, OSASetScriptInfo sets a value indicating the number of times the script data has been modified since it was created or passed to [OSALoad](#). Some scripting components may provide additional constants.

Special Considerations

Although you can specify kOSAScriptIsModified when you are using the Object REXX component without generating an error, the current version of Object REXX does not actually set a value for the count of changes since the script data was loaded or created.

OSASetScriptInfo - Override Policy

A scripting component must override this method.

OSASetScriptInfo - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSASetSendProc

OSASetSendProc - Syntax

This method specifies a send function that a scripting component should use instead of the OSA Event Manager's AESend function when sending OSA events.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSASendUPP    sendProc;
ODSLong       refCon;
OSAError      rc;          /* Return code. */

rc = OSASetSendProc(sendProc, refCon);
```

OSASetSendProc Parameter - sendProc

sendProc ([OSASendUPP](#)) - input
A pointer to the send function to be used when sending OSA events.

OSASetSendProc Parameter - refCon

refCon ([ODSLong](#)) - input
A reference constant to be associated with the send function.

OSASetSendProc Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetSendProc - Parameters

sendProc ([OSASendUPP](#)) - input

A pointer to the send function to be used when sending OSA events.

refCon ([ODSLong](#)) - input

A reference constant to be associated with the send function.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetSendProc - Override Policy

A scripting component which sets `kOSASupportsAESending` in the *componentFlags* field of the component description record must override this method.

OSASetSendProc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Override Policy](#)

[Glossary](#)

OSASetResumeDispatchProc

OSASetResumeDispatchProc - Syntax

This method sets the resume dispatch function called by a scripting component during execution of a scripting component's continue

statement or its equivalent.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAEventHandlerUPP    resumeDispatchProc;
ODSLong               refCon;
OSAError              rc;                                /* Return code. */

rc = OSASetResumeDispatchProc(resumeDispatchProc,
                               refCon);
```

OSASetResumeDispatchProc Parameter - resumeDispatchProc

resumeDispatchProc ([OSAEventHandlerUPP](#)) - input

A pointer to a resume dispatch function. This parameter can be set to the following value:

kOSAUseStandardDispatch	Tells the OSA Event Manager that the processing of the OSA event is complete and does not need to be dispatched.
kOSANoDispatch	Causes the OSA Event Manager to dispatch the event using standard OSA event dispatching.

OSASetResumeDispatchProc Parameter - refCon

refCon ([ODSLong](#)) - input

A reference constant. Specify kOSADontUsePhac in this parameter and kOSAUseStandardDispatch in the *resumeDispatchProc* parameter to request standard OSA event dispatching excluding the special handler dispatch table.

kOSADontUsePhac	When kOSAUseStandardDispatch is specified in the <i>resumeDispatchProc</i> parameter, this flag requests standard OSA event dispatching excluding the special handler dispatch table.
-----------------	---

OSASetResumeDispatchProc Return Value - rc

rc ([OSAError](#)) - returns

Return code.

noErr	No error.
errOSASystemError	General scripting system error.

OSASetResumeDispatchProc - Parameters

resumeDispatchProc ([OSAEventHandlerUPP](#)) - input

A pointer to a resume dispatch function This parameter can be set to the following value:

kOSAUseStandardDispatch

Tells the OSA Event Manager that the processing of the OSA event is complete and does not need to be dispatched.

kOSANoDispatch

Causes the OSA Event Manager to dispatch the event using standard OSA event dispatching.

refCon ([ODSLong](#)) - input

A reference constant. Specify kOSADontUsePhac in this parameter and kOSAUseStandardDispatch in the *resumeDispatchProc* parameter to request standard OSA event dispatching excluding the special handler dispatch table.

kOSADontUsePhac

When kOSAUseStandardDispatch is specified in the *resumeDispatchProc* parameter, this flag requests standard OSA event dispatching excluding the special handler dispatch table.

rc ([OSAError](#)) - returns

Return code.

noErr

No error.

errOSASystemError

General scripting system error.

OSASetResumeDispatchProc - Remarks

This method sets the resume dispatch function that the specified instance of a scripting component calls during execution of a scripting component's continue statement or its equivalent. The resume dispatch function should dispatch the event to the application's standard handler for that event.

If you are using a general handler (similar to that in the figure in section [Using a Script Context to Handle an OSA Event](#)) for preliminary processing of OSA events and if you can rely on standard OSA event dispatching to dispatch the event correctly, you do not need to provide a resume dispatch function. Instead, you can specify kOSAUseStandardDispatch as the value of the *resumeDispatchProc* parameter and the constant kOSADontUsePhac as the value of the *refCon* parameter. This causes the OSA Event Manager to use standard OSA event dispatching except that it bypasses your application's special handler dispatch table and thus will not call your general OSA event handler recursively.

OSASetResumeDispatchProc - Override Policy

A scripting component which sets kOSASupportsEventHandling in the *componentFlags* field of the component description record must override this method.

OSASetResumeDispatchProc - Topics

Class:

OSAScriptingComponent

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Override Policy](#)

[Glossary](#)

OSAStartRecording

OSAStartRecording - Syntax

This method turns on OSA event recording and records subsequent OSA events in a compiled script.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      *pcompiledScriptToModifyID;
OSAError    rc;          /* Return code. */

rc = OSAStartRecording(pcompiledScriptToModifyID);
```

OSAStartRecording Parameter - pcompiledScriptToModifyID

pcompiledScriptToModifyID (OSAID *) - output
The script ID for the compiled script in which to record.

OSAStartRecording Return Value - rc

rc (OSAError) - returns
Return code.

noErr	No error.
errAERecordingIsAlreadyOn	Attempt to turn recording on when it is already on for a recording process.
errOSASystemError	General scripting system error.
errOSAInvalidID	Invalid script ID.

OSAStartRecording - Parameters

pcompiledScriptToModifyID (OSAID *) - output
The script ID for the compiled script in which to record.

rc (OSAError) - returns
Return code.

noErr	No error.
-------	-----------

errAERecordingIsAlreadyOn Attempt to turn recording on when it is already on for a recording process.
errOSASystemError General scripting system error.
errOSAInvalidID Invalid script ID.

OSAStartRecording - Remarks

This method turns on OSA event recording. Subsequent OSA events are recorded-that is, appended to any existing statements-in the compiled script specified by the *pcompiledScriptToModifyID* parameter. If the source data for the compiled script is currently displayed in a script editor's window, the script editor's handler for the Recorded Text event should display each new statement in the window as it is recorded. Users should not be able to change a script that is open in a script editor window while it is being recorded into.

To record into a new compiled script, pass the constant kOSANullScript in the *pcompiledScriptToModifyID* parameter. The scripting component should respond by creating a new compiled script and recording into that.

Special Considerations

The generic scripting component uses its default scripting component to create and record into a new compiled script.

For more information about the default scripting component associated with any instance of the generic scripting component, see [Generic Scripting Component Routines](#).

OSAStartRecording - Override Policy

A scripting component which sets kOSASupportsRecording in the *componentFlags* field of the component description record must override this method.

OSAStartRecording - Related Methods

Related Methods

- [OSAScriptingComponent::OSAStopRecording](#)
-

OSAStartRecording - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Related Methods](#)
[Glossary](#)

OSAStopRecording

OSAStopRecording - Syntax

This method turns off OSA event recording.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      compiledScriptID;
OSAError    rc;          /* Return code. */

rc = OSAStopRecording(compiledScriptID);
```

OSAStopRecording Parameter - compiledScriptID

compiledScriptID ([OSAID](#)) - input
A script ID for the compiled script into which OSA events are being recorded.

OSAStopRecording Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSAInvalidID	Invalid script ID.

OSAStopRecording - Parameters

compiledScriptID ([OSAID](#)) - input
A script ID for the compiled script into which OSA events are being recorded.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.

OSAStopRecording - Remarks

This method turns off recording. If the script is not currently open in a script editor window, the *pcompiledScriptToModifyID* parameter supplied to [OSASStartRecording](#) is then augmented to contain the newly recorded statements. If the script is currently open in a script editor window, the script data that corresponds to the *pcompiledScriptToModifyID* parameter supplied to [OSASStartRecording](#) *formt=extonly* is updated continuously until the client application calls [OSAStopRecording](#).

If the compiled script identified by the script ID in the *compiledScriptID* parameter is not being recorded into or recording is not currently on, this method returns noErr.

OSAStopRecording - Override Policy

A scripting component which sets *kOSASupportsRecording* in the *componentFlags* field of the component description record must override this method.

OSAStopRecording - Related Methods

Related Methods

- [OSAScriptingComponent::OSASStartRecording](#)
-

OSAStopRecording - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Related Methods](#)
[Glossary](#)

OSAStore

OSAStore - Syntax

This method returns a handle to script data in the form of a storage descriptor record.

```

#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OSAAPI
#define INCL_OSA
#include <os2.h>

OSAID      scriptID;
DescType    desiredType;
ODSLong     modeFlags;
AEDesc      *presultingScriptData;
OSAError    rc;          /* Return code. */

rc = OSASStore(scriptID, desiredType, modeFlags,
               presultingScriptData);

```

OSASStore Parameter - scriptID

scriptID ([OSAID](#)) - input
 The script ID for the script data for which to obtain a data handle.

OSASStore Parameter - desiredType

desiredType ([DescType](#)) - input
 The desired type of the descriptor record to be returned.

This parameter is set to a specific scripting component subtype value for a component-specific storage descriptor record or to the following for a generic storage descriptor record:

typeOSAGenericStorage
 Store the script data in the form used by a generic storage descriptor record.

OSASStore Parameter - modeFlags

modeFlags ([ODSLong](#)) - input
 Information used by individual scripting components.

kOSAModeNull
 No flags are set.

kOSAModePreventGetSource
 Only the minimum script data required to run the script should be returned. (In this case, the script data returned is not identical to the compiled script data and cannot be used to generate source data.)

kOSAModeDontStoreParent
 If the *scriptID* parameter identifies a script context, setting this value stores the script context without storing its parent context.

OSASStore Parameter - presultingScriptData

presultingScriptData ([AEDesc *](#)) - output
The resulting descriptor record.

OSAStore Return Value - rc

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSAInvalidID	Invalid script ID was specified.
errOSABadStorageType	Desired type is not supported by this scripting component.

OSAStore - Parameters

scriptID ([OSAID](#)) - input
The script ID for the script data for which to obtain a data handle.

desiredType ([DescType](#)) - input
The desired type of the descriptor record to be returned.

This parameter is set to a specific scripting component subtype value for a component-specific storage descriptor record or to the following for a generic storage descriptor record:

typeOSAGenericStorage	Store the script data in the form used by a generic storage descriptor record.
-----------------------	--

modeFlags ([ODSLong](#)) - input
Information used by individual scripting components.

kOSAModeNull	No flags are set.
kOSAModePreventGetSource	Only the minimum script data required to run the script should be returned. (In this case, the script data returned is not identical to the compiled script data and cannot be used to generate source data.)
kOSAModeDontStoreParent	If the <i>scriptID</i> parameter identifies a script context, setting this value stores the script context without storing its parent context.

presultingScriptData ([AEDesc *](#)) - output
The resulting descriptor record.

rc ([OSAError](#)) - returns
Return code.

noErr	No error.
errOSASystemError	General scripting system error.
errOSAInvalidID	Invalid script ID was specified.
errOSABadStorageType	Desired type is not supported by this scripting component.

OSAStore - Remarks

This method writes script data to a descriptor record so that the data can later be saved in a file. You can then reload the data for the descriptor record as a compiled script (although possibly with a different script ID) by passing the descriptor record to the [OSALoad](#) method.

If you want the returned script data to be as small as possible and you are sure that you will not need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the *modeFlags* parameter. If the *scriptID* parameter identifies a script context and you do not want the returned script data to include the associated parent context, specify the `kOSAModeDontStoreParent` flag in the *modeFlags* parameter.

The desired type is either `typeOSAGenericStorage` (for a generic storage descriptor record) or a specific scripting component subtype value (for a component-specific storage descriptor record). All scripting components are expected to support a desired type of `typeOSAGenericStorage`. When the desired type is passed, the scripting component must call [OSAAddStorageType](#) to generate a generic storage descriptor.

To store either a generic storage descriptor record or a component-specific storage descriptor record with your application's resources, use 'scpt' as the resource type. The generic scripting component subtype, the generic storage descriptor type, and the resource type for stored script data all have the same value, even though they serve different purposes.

```
#define kOSAGenericScriptingComponentSubtype 0x74706373 /* "scpt" */
#define kOSAScriptResourceType                kOSAGenericScriptingComponentS
#define typeOSAGenericStorage                 kOSAScriptResourceType
```

Scripting components must also accept their own component subtype in the *desiredType* parameter. In some cases, the scripting component is not required to generate a generic storage descriptor; however, it is recommended that scripting components create a generic storage descriptor if it will not interfere with proper interpretation of the script during execution. Any other value in *desiredType* parameter should be treated as an error and return `errOSABadStorageType`.

Object REXX Notes

The *desiredType* parameter must be of type `typeOREXStorage` or `typeOSAGenericStorage`.

OSAStore - Override Policy

A scripting component must override this method.

OSAStore - Topics

Class:

[OSAScriptingComponent](#)

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Override Policy](#)
[Glossary](#)

OSATerminology

Class Definition File: OSATERM.IDL

Class Hierarchy

SOMObject

OSATerminology

Description

The OSATerminology class provides methods that scripting components use to read and merge terminology resources. This class is unique to OS/2.

This class treats the combination of country and code page settings as a language.

Methods

The following list shows the methods defined by the OSATerminology class:

- [GetAETE](#)
- [GetAEUT](#)
- [GetSCSZFlags](#)
- [ListApplications](#)
- [PutAETE](#)
- [PutSCSZFlags](#)
- [QueryExecutableFileName](#)
- [RegisterApplication](#)
- [UnregisterApplication](#)

Overridden Methods

There are currently no methods overridden by the OSATerminology class.

GetAETE (OS/2)

GetAETE (OS/2) - Syntax

This method returns the **aete** resource of an application.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char          *pszApp_ID;
ODBoolean     launchIfNeeded;
AEDescList    *ptheAETEs;
ODULong       Reserved1;
ODULong       Reserved2;
OSAError      rc;          /* Return code. */

rc = GetAETE(pszApp_ID, launchIfNeeded, ptheAETEs,
             Reserved1, Reserved2);
```

GetAETE (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

GetAETE (OS/2) Parameter - launchIfNeeded

launchIfNeeded (ODBoolean) - input

A flag indicating the amount of processing to be done for applications which have the kLaunchToGetTerminology bit set in their **scsz** resource. If this bit is not set, this parameter is ignored. If the bit is set, this parameter is interpreted in the following manner:

TRUE

The application is to be launched to get its terminology.

FALSE

Only the **aete** resource from the registration data base is to be returned. The application is not to be launched.

GetAETE (OS/2) Parameter - ptheAETEs

ptheAETEs (AEDescList *) - output

A list of **aete** resources. (Most application return only a single item in this list.)

If the application's **scsz** resource indicates that the application must be launched to obtain the **aete** resource and the *launchIfNeeded* parameter is TRUE, the application is launched and sent a Get AETE event. In this case, a list of **aete** resources can be returned.

GetAETE (OS/2) Parameter - Reserved1

Reserved1 (ODULong) - input

Reserved value.

GetAETE (OS/2) Parameter - Reserved2

Reserved2 (ODULong) - input

Reserved value.

GetAETE (OS/2) Return Value - rc

rc (OSAEError) - returns

Return code.

errAEResourceNotFound

The requested resource was not found in the registration data base.

errAEBadParm

	One or more of the parameters passed in were invalid.
errAEAppNotFound	The specified application or part handler was not found in the registration data base.
errOSASystemError	A general scripting system error occurred.

GetAETE (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

launchIfNeeded (ODBoolean) - input

A flag indicating the amount of processing to be done for applications which have the kLaunchToGetTerminology bit set in their **scsz** resource. If this bit is not set, this parameter is ignored. If the bit is set, this parameter is interpreted in the following manner:

TRUE

The application is to be launched to get its terminology.

FALSE

Only the **aete** resource from the registration data base is to be returned. The application is not to be launched.

ptheAETEs (AEDescList *) - output

A list of **aete** resources. (Most application return only a single item in this list.)

If the application's **scsz** resource indicates that the application must be launched to obtain the **aete** resource and the *launchIfNeeded* parameter is TRUE, the application is launched and sent a Get AETE event. In this case, a list of **aete** resources can be returned.

Reserved1 (ODULong) - input

Reserved value.

Reserved2 (ODULong) - input

Reserved value.

rc (OSAEError) - returns

Return code.

errAEResourceNotFound

The requested resource was not found in the registration data base.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

errOSASystemError

A general scripting system error occurred.

GetAETE (OS/2) - Example Code

This example gets the **aete** resource for an application and copies the data into a local buffer.

```

OSATerminology *term;
AEDescList      desclist;
AEDesc          desc = {typeNull, NULL};
AEKeyword       aekey = NULL;
PBYTE          pBuffer = NULL;
Size            dataSize;
DescType        typeCode;
ODSLong         count = 0;
ODULong         reserved = 0;
USHORT          i = 0;
OSAEError       rc;

```

```

/* Create an instance of the OSATerminology class. */
term = new OSATerminology;

/* Get the AETE for an application. */
rc = term->GetAETE(ev, "My OSA App", /* Name of application */
                  FALSE, /* Don't launch app to get the AETE */
                  &desclist, /* ptr to where AETE(s) are copied */
                  reserved, reserved);

/* Find out how many AETE's in the list. */
AECountItems(&desclist, &count);

/* For each aete: */
/*   copy it into a local buffer */
/*   do something with it */
/*   free buffer */
for(i = 1; i <= count; i++)
{
    rc = AEGetNthDesc(&desclist, i, typeAETE, &aekey, &desc);

    /* Get the size of AETE. */
    AESizeOfDescData(&desc, &dataSize);

    /* Allocate storage for AETE. */
    pBuffer = (PBYTE) malloc(dataSize);

    /* Now copy data from desc into our local buffer. */
    AEGetDescData(&desc, &typeCode, (Ptr) pBuffer, dataSize, &dataSize);
    .
    .
    .

    free(pBuffer);
}

AEDisposeDesc(&desclist);

```

GetAETE (OS/2) - Topics

Class:

OSATerminology

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

GetAEUT (OS/2)

GetAEUT (OS/2) - Syntax

This method returns the system's **aeut** resource that defines the events and classes in the *OSA Event Registry: Standard Suites* .

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

AEDesc      *ptheAEUT;
ODULong      Reserved1;
ODULong      Reserved2;
OSAError      rc;          /* Return code. */

rc = GetAEUT(ptheAEUT, Reserved1, Reserved2);
```

GetAEUT (OS/2) Parameter - ptheAEUT

ptheAEUT (**AEDesc** *) - output
The system's **aeut** resource.

GetAEUT (OS/2) Parameter - Reserved1

Reserved1 (**ODULong**) - input
Reserved value.

GetAEUT (OS/2) Parameter - Reserved2

Reserved2 (**ODULong**) - input
Reserved value.

GetAEUT (OS/2) Return Value - rc

rc (**OSAError**) - returns
Return code.

errAEBadParm One or more of the parameters passed in were invalid.
errOSASystemError A general scripting error occurred.

GetAEUT (OS/2) - Parameters

ptheAEUT ([AEDesc](#) *) - output
The system's **aeut** resource.

Reserved1 ([ODULong](#)) - input
Reserved value.

Reserved2 ([ODULong](#)) - input
Reserved value.

rc ([OSAError](#)) - returns
Return code.

errAEBadParm One or more of the parameters passed in were invalid.
errOSASystemError A general scripting error occurred.

GetAEUT (OS/2) - Example Code

This example gets the system **aeut** resource and copies the data into a local buffer.

```
OSATerminology *term;  
AEDesc                desc = {typeNull, NULL};  
PBYTE                pBuffer = NULL;  
Size                  dataSize;  
DescType              typeCode;  
ODULong               reserved = 0;  
OSAError              rc;  
  
/* Create an instance of the OSATerminology class. */  
term = new OSATerminology;  
  
/* Get the system AEUT. */  
rc = term->GetAEUT(ev, &desc, reserved, reserved);  
  
/* Get the size of AEUT. */  
AESizeOfDescData(&desc, &dataSize);  
  
/* Allocate storage for AEUT. */  
pBuffer = (PBYTE) malloc(dataSize);  
  
/* Now, copy data from desc into our local buffer. */  
AEGetDescData(&desc, &ano, typeCode, (Ptr) pBuffer, dataSize, &dataSize);  
  
AEDisposeDesc(&desc);
```

GetAEUT (OS/2) - Topics

Class:
OSATERMINOLOGY

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

GetSCSZFlags (OS/2)

GetSCSZFlags (OS/2) - Syntax

This method returns the scripting size flags field of an application's **scsz** resource.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
ODULong   *pscszFlags;
OSAError   rc;      /* Return code. */

rc = GetSCSZFlags(pszApp_ID, pscszFlags);
```

GetSCSZFlags (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input
A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

GetSCSZFlags (OS/2) Parameter - pscszFlags

pscszFlags (ODULong *) - output
The application's **scsz** flags. This parameter can be set to any of the following flags:

- kLaunchToGetTerminology
This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is

launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the applications **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kDontFindAppBySignature

This bit is reserved and must be set to 1.

kAlwaysSendSubject

When this bit is set, scripting components and other applications which sent events to this application must include a subject attribute in the event.

GetSCSZFlags (OS/2) Return Value - rc

rc ([OSAEError](#)) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEResourceNotFound

The requested resource was not found in the registration data base.

GetSCSZFlags (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

pscszFlags ([ODULong](#) *) - output

The application's **scsz** flags. This parameter can be set to any of the following flags:

kLaunchToGetTerminology

This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the applications **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kDontFindAppBySignature

This bit is reserved and must be set to 1.

kAlwaysSendSubject

When this bit is set, scripting components and other applications which sent events to this application must include a subject attribute in the event.

rc ([OSAEError](#)) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEResourceNotFound

The requested resource was not found in the registration data base.

GetSCSZFlags (OS/2) - Example Code

This example gets the **scsz** flags for an application.

```

OSAterminology *term;
ODULong      flags;
OSAError      rc;

/* Create an instance of the OSAterminology class. */
term = new OSAterminology;

/* Get the SCSZ flags for an OSA Application. */
rc = term->PutSCSZFlags(ev, "My OSA App", &flags);

```

GetSCSZFlags (OS/2) - Topics

Class:
OSAterminology

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

ListApplications (OS/2)

ListApplications (OS/2) - Syntax

This method returns a list of registered applications.

```

#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszAppName;
ODULong   *psize;
ODULong   AppType;
OSAError   rc;          /* Return code. */

rc = ListApplications(pszAppName, psize, AppType);

```

ListApplications (OS/2) Parameter - pszAppName

pszAppName (char *) - output

A pointer to a buffer containing a list of null-terminated strings specifying the descriptive names for each registered application. These names can be used with the [QueryExecutableFileName](#) method to obtain the executable file name.

The caller can determine the size of the buffer required to hold all of the descriptive names by calling this method with *pszAppName* parameter set to NULL. In this case, the required buffer size is returned in the *psize* parameter. A buffer of this size can be allocated and its address passed on a second call to ListApplications.

ListApplications (OS/2) Parameter - psize

psize ([ODULong *](#)) - in/out

On input, this parameter specifies the size, in bytes, of the buffer allocated by the caller. On output, the number of bytes written to the buffer is returned.. An extra NULL byte, included in this count, follows the terminating NULL byte of the last name string. The space for this extra byte is returned when ListApplications is called to determine the required buffer size by passing a NULL buffer pointer.

ListApplications (OS/2) Parameter - AppType

AppType ([ODULong](#)) - input

The type of applications to be listed.

OSA_APPLICATION

List PM applications only.

OSA_PARTHANDLER

List OpenDoc part handlers only.

OSA_BOTH

List both PM applications and OpenDoc part handlers.

ListApplications (OS/2) Return Value - rc

rc ([OSAEError](#)) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

errAEBufferTooSmall

The size of the buffer is not large enough to hold the data to be returned.

ListApplications (OS/2) - Parameters

pszAppName ([char *](#)) - output

A pointer to a buffer containing a list of null-terminated strings specifying the descriptive names for each registered application. These names can be used with the [QueryExecutableFileName](#) method to obtain the executable file name.

The caller can determine the size of the buffer required to hold all of the descriptive names by calling this method with *pszAppName* parameter set to NULL. In this case, the required buffer size is returned in the *psize* parameter. A buffer of this size can be allocated and its address passed on a second call to ListApplications.

psize (ODULong *) - in/out

On input, this parameter specifies the size, in bytes, of the buffer allocated by the caller. On output, the number of bytes written to the buffer is returned.. An extra NULL byte, included in this count, follows the terminating NULL byte of the last name string. The space for this extra byte is returned when ListApplications is called to determine the required buffer size by passing a NULL buffer pointer.

AppType (ODULong) - input

The type of applications to be listed.

OSA_APPLICATION

List PM applications only.

OSA_PARTHANDLER

List OpenDoc part handlers only.

OSA_BOTH

List both PM applications and OpenDoc part handlers.

rc (OSAEError) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

errAEBufferTooSmall

The size of the buffer is not large enough to hold the data to be returned.

ListApplications (OS/2) - Example Code

This example prints a list of registered OSA-aware applications.

```
OSATerminology *term;
```

```
PSZ             pszBuffer = NULL;
```

```
PSZ             pszTemp = NULL;
```

```
ULONG          size = 0;
```

```
ULONG          appType;
```

```
OSAEError       rc;
```

```
/* Create an instance of the OSATerminology class. */
```

```
term = new OSATerminology;
```

```
/* Set appType to look for applications. */
```

```
/* (could also look for parts, OSA_PARTHANDLER, or both, OSA_ALL) */
```

```
appType = OSA_APPLICATION;
```

```
/* First, call list applications to get the required buffer size. */
```

```
rc = term->ListApplications(ev, NULL, &size, appType);
```

```
if(size)
```

```
{
```

```
    /* Allocate required buffer. */
```

```
    pszBuffer = (char *) malloc(size);
```

```
    /* Now, call list applications with our buffer. */
```

```
    term->ListApplications(ev, pszBuffer, &size, appType);
```

```
}
```

```
pszTemp = pszBuffer; /* Save ptr for free. */
```

```
/* Print out the name of each application returned. */
```

```
while(strlen(pszBuffer))
{
    printf("App Name = %s\n",pszBuffer);
    pszBuffer = pszBuffer + (strlen(pszBuffer) + 1);
}

free(pszTemp);
```

ListApplications (OS/2) - Topics

Class:
 OSATerminology

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Glossary](#)

PutAETE (OS/2)

PutAETE (OS/2) - Syntax

This method writes the **aete** resource for the application to the registration data base.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
AEDesc    *ptheAETE;
ODULong    Reserved1;
ODULong    Reserved2;
OSAError    rc;          /* Return code. */

rc = PutAETE(pszApp_ID, ptheAETE, Reserved1,
             Reserved2);
```

PutAETE (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

PutAETE (OS/2) Parameter - ptheAETE

ptheAETE ([AEDesc *](#)) - input
The **aete** resource to write to the registration data base.

PutAETE (OS/2) Parameter - Reserved1

Reserved1 ([ODULong](#)) - input
Reserved value.

PutAETE (OS/2) Parameter - Reserved2

Reserved2 ([ODULong](#)) - input
Reserved value.

PutAETE (OS/2) Return Value - rc

rc ([OSAEError](#)) - returns
Return code.

errAEBadParm
One or more of the parameters passed in were invalid.

errAEAppNotFound
The specified application or part handler was not found in the registration data base.

PutAETE (OS/2) - Parameters

pszApp_ID (char *) - input
A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

ptheAETE ([AEDesc *](#)) - input
The **aete** resource to write to the registration data base.

Reserved1 ([ODULong](#)) - input
Reserved value.

Reserved2 ([ODULong](#)) - input
Reserved value.

rc ([OSAEError](#)) - returns
Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

PutAETE (OS/2) - Example Code

This example puts **aete** resource for the application in the registration data base.

```
OSATerminology *term;
AEDesc          desc = {typeNull, NULL};
PBYTE           pBuffer;
ODULong         size;
ODULong         reserved = 0;
OSAEError       rc;

/* Create an instance of the OSATerminology class. */
term = new OSATerminology;

        .
        . /* Init pBuffer with an AETE with size 'size'. */
        .

/* Create a descriptor for the AETE. */
AECreatDesc(typeAETE, pBuffer, size, &desc);

/* Put the AETE for an application. */
rc = term->PutAETE(ev, "My OSA App",          /* name of application */
                  &desc,                    /* descriptor containing AETE */
                  reserved, reserved);
```

PutAETE (OS/2) - Topics

Class:

[OSATerminology](#)

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

PutSCSZFlags (OS/2)

PutSCSZFlags (OS/2) - Syntax

This method writes the **scsz** resource to the registration data base.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
ODULong   scszFlags;
OSAError   rc;          /* Return code. */

rc = PutSCSZFlags(pszApp_ID, scszFlags);
```

PutSCSZFlags (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

PutSCSZFlags (OS/2) Parameter - scszFlags

scszFlags (ODULong) - input

The **scsz** flags to write to the registration data base.

kLaunchToGetTerminology

This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the application's **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kDontFindAppBySignature

This bit is reserved and must be set to 1.

kAlwaysSendSubject

When this bit is set, scripting components and other applications that sent events to this application must include a subject attribute in the event.

PutSCSZFlags (OS/2) Return Value - rc

rc (OSAError) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

PutSCSZFlags (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

scszFlags (ODULong) - input

The **scsz** flags to write to the registration data base.

kLaunchToGetTerminology

This bit indicates whether the **aete** resource of an application is dynamic and configured by the application when it is launched (for example, the application supports add-on tools which are optionally loaded during initialization).

If this bit is set, a scripting component or application must launch the application and issue a Get AETE event to obtain the application's **aete** resource. Otherwise, the application's **aete** resource can be read directly.

kDontFindAppBySignature

This bit is reserved and must be set to 1.

kAlwaysSendSubject

When this bit is set, scripting components and other applications that sent events to this application must include a subject attribute in the event.

rc (OSAEError) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

PutSCSZFlags (OS/2) - Remarks

Some events in the registry do not identify the object to which an event is to be sent in the direct parameter of the event. For these events, the direct parameter is data to operate on, rather than an object specifier for the object to receive the event (for example, the Count Elements and Open events in the Core suites).

Applications, such as the OpenDoc document shell, that dispatch events by first resolving the object specifier cannot properly process these events. Scripting components must add a subject attribute (keySubjectAttr) to these events; therefore, if the direct parameter is not an object specifier, a subject attribute should be included in the event which contains the object specifier for the target object.

If the direct parameter is not an object specifier, server applications should check for the presence of a subject attribute during dispatching.

The kAlwaysSendSubject bit in the *scszFlags* parameter is defined to give applications more control over which handlers are used when an event is processed. If server applications want the target object to always handle an event, regardless of what the direct object of the event contains, these applications must set the kAlwaysSendSubject bit. Even if the direct parameter of the event is an object specifier, all events sent to these applications must include a subject attribute which identifies the target of the event.

Object REXX sends a subject attribute under the following two conditions regardless of whether the kAlwaysSendSubject bit is set or whether the application has an **scsz** resource:

- *Direct parameter optional.* If the direct parameter of an event is optional and is omitted from a command, Object REXX always sends a subject attribute with the object specifier for the application as the target (null object specifier).
 - *Direct parameter required.* If the direct parameter of an event is required and is omitted from a command, Object REXX always sends a subject attribute with an object specifier for the application as the target (null object specifier), as well as a direct parameter with the same object specifier (the direct parameter from the tell statement).
-

PutSCSZFlags (OS/2) - Example Code

This example sets the **scsz** flags for an OSA-aware application.

```
OSATerminology *term;
ODULong        flags;
OSAError        rc;

/* Create an instance of the OSATerminology class. */
term = new OSATerminology;

/* Set the SCSZ flags for an OSA Application. */
flags = 0x00000000 | kAlwaysSendSubject;
rc = term->PutSCSZFlags(ev, "My OSA App", flags);
```

PutSCSZFlags (OS/2) - Topics

Class:

OSATerminology

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

QueryExecutableFileName (OS/2)

QueryExecutableFileName (OS/2) - Syntax

This method returns the executable file name of the specified application.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
char      *pszApp_ExecutableName;
ODULong   *psize;
OSAError   rc;                      /* Return code. */

rc = QueryExecutableFileName(pszApp_ID, pszApp_ExecutableName,
                             psize);
```

QueryExecutableFileName (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler whose executable file name is to be returned.

QueryExecutableFileName (OS/2) Parameter - pszApp_Executable

pszApp_ExecutableName (char *) - output

A pointer to a buffer allocated by the caller containing a null-terminated ASCII string specifying the executable file name of the requested application or part handler.

The caller can determine the size of the buffer required to hold the executable file name by calling this method with *pszApp_ExecutableName* parameter set to KODNull. In this case, the *psize* parameter returns the required buffer size. A buffer of this size can then be allocated and its address passed on a second call to QueryExecutableFileName.

QueryExecutableFileName (OS/2) Parameter - psize

psize (ODULong *) - in/out

On input, this parameter specifies the size, in bytes, of the buffer allocated by the caller. On output, the number of bytes written to the buffer, including the terminating null byte is returned.

QueryExecutableFileName (OS/2) Return Value - rc

rc (OSAEError) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

QueryExecutableFileName (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler whose executable file name is to be returned.

pszApp_ExecutableName (char *) - output

A pointer to a buffer allocated by the caller containing a null-terminated ASCII string specifying the executable file name of the requested application or part handler.

The caller can determine the size of the buffer required to hold the executable file name by calling this method with *pszApp_ExecutableName* parameter set to KODNull. In this case, the *psize* parameter returns the required buffer size. A buffer of this size can then be allocated and its address passed on a second call to QueryExecutableFileName.

psize ([ODULong *](#)) - in/out

On input, this parameter specifies the size, in bytes, of the buffer allocated by the caller. On output, the number of bytes written to the buffer, including the terminating null byte is returned.

rc ([OSAError](#)) - returns

Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

QueryExecutableFileName (OS/2) - Example Code

This example prints the executable file name of an application.

```
OSATerminology *term;
PSZ             pszBuffer = NULL;
ULONG           size = 0;
OSAError        rc;

/* Create an instance of the OSATerminology class. */
term = new OSATerminology;

/* First find out the buffer size required by passing a NULL buffer. */
term->QueryExecutableFileName(ev, "My OSA App", NULL, &size);

if(size)
{
    /* Allocate required storage. */
    pszBuffer = (char *) malloc(size);

    /* Call it again with allocated buffer. */
    term->QueryExecutableFileName(ev, "My OSA App", pszBuffer, &size);

    /* Print out the executable name. */
    printf("App_EXE = %s\n", pszBuffer);

    free (pszBuffer);
}
```

QueryExecutableFileName (OS/2) - Topics

Class:

[OSATerminology](#)

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

RegisterApplication (OS/2)

RegisterApplication (OS/2) - Syntax

This method creates an initial entry in the registration data base for the application or part handler. Additional resources, such as **aete** or **scsz**, can then be added for this application.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
char      *pszApp_ExecutableName;
ODULong   AppType;
OSAError   rc;                      /* Return code. */

rc = RegisterApplication(pszApp_ID, pszApp_ExecutableName,
                        AppType);
```

RegisterApplication (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

RegisterApplication (OS/2) Parameter - pszApp_ExecutableName

pszApp_ExecutableName (char *) - input

A pointer to a null-terminated ASCII string containing the executable file name of the application. If the file name is not fully qualified, the PATH and LIBPATH variables in CONFIG.SYS file must be set up properly to locate the application or part handler.

RegisterApplication (OS/2) Parameter - AppType

AppType (ODULong) - input

The type of application to be registered.

OSA_APPLICATION

A PM application

OSA_PARTHANDLER

An OpenDoc part handler

RegisterApplication (OS/2) Return Value - rc

rc ([OSAError](#)) - returns
Return code.

errAEBadParm

One or more of the parameters passed in are invalid.

errAEAppAlreadyInstalled

The application or part handler is already in the registration data base.

RegisterApplication (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler.

pszApp_ExecutableName (char *) - input

A pointer to a null-terminated ASCII string containing the executable file name of the application. If the file name is not fully qualified, the PATH and LIBPATH variables in CONFIG.SYS file must be set up properly to locate the application or part handler.

AppType ([ODULong](#)) - input

The type of application to be registered.

OSA_APPLICATION

A PM application

OSA_PARTHANDLER

An OpenDoc part handler

rc ([OSAError](#)) - returns
Return code.

errAEBadParm

One or more of the parameters passed in are invalid.

errAEAppAlreadyInstalled

The application or part handler is already in the registration data base.

RegisterApplication (OS/2) - Example Code

This example registers an application in the OSA registration data base.

```
OSATerminology *term;
```

```
OSAError      rc;
```

```
/* Create an instance of the OSATerminology class. */
```

```
term = new OSATerminology;
```

```
/* Register my application in the OSA registration data base. */
```

```
rc = term->RegisterApplication(ev,  
                               "My OSA App",      /* descriptive app name */  
                               "myapp.exe",       /* app's exe name          */  
                               OSA_APPLICATION); /* app type.          */
```

```
/* Now, we can also use PutAETE and PutSCSZ methods. */
```

.
.
.

RegisterApplication (OS/2) - Topics

Class:

OSATerminology

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

UnregisterApplication (OS/2)

UnregisterApplication (OS/2) - Syntax

This method removes all entries for the application or part handler from the registration data base.

```
#define INCL_ODAPI
#define INCL_OSACOMPONENT
#define INCL_OAAPI
#define INCL_OSA
#include <os2.h>

char      *pszApp_ID;
OSAError  rc;          /* Return code. */

rc = UnregisterApplication(pszApp_ID);
```

UnregisterApplication (OS/2) Parameter - pszApp_ID

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler to be removed.

UnregisterApplication (OS/2) Return Value - rc

rc ([OSAEError](#)) - returns
Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

UnregisterApplication (OS/2) - Parameters

pszApp_ID (char *) - input

A pointer to a null-terminated ASCII string containing the descriptive name of the application or part handler to be removed.

rc ([OSAEError](#)) - returns
Return code.

errAEBadParm

One or more of the parameters passed in were invalid.

errAEAppNotFound

The specified application or part handler was not found in the registration data base.

UnregisterApplication (OS/2) - Example Code

This example unregisters an application with the OSA registration data base.

```
OSATerminology *term;
```

```
OSAEError      rc;
```

```
/* Create an instance of the OSATerminology class. */
```

```
term = new OSATerminology;
```

```
/* Remove my application from the OSA registration data base; */
```

```
/* this removes my app's AETE and SCSZ information as well. */
```

```
rc = term->UnregisterApplication(ev, "My OSA App");
```

UnregisterApplication (OS/2) - Topics

Class:

[OSATerminology](#)

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Example Code](#)

[Glossary](#)

Data Types

The following data types are used in the Open Scripting Architecture APIs. They are listed in alphabetic order.

AEAddressDesc

Descriptor record containing address data.

```
typedef AEDesc AEAddressDesc;
```

The address in the address descriptor record can be specified as one of the following type:

Descriptor Type	Value	Description
typePID	PID	Process ID

AEArrayData

Data for an OSA event array.

```
union AEArrayData {  
    short      kAEDataArray[1];  
    char       kAEPackedArray[1];  
    Handle     kAEHandleArray[1];  
    AEDesc     kAEDescArray[1];  
    AEKeyDesc  kAEKeyDescArray[1];  
} AEArrayData;
```

AEArrayData Field - kAEDataArray[1]

kAEDataArray[1] (short)

Array items consist of data of the same size and same type, and are aligned on word boundaries.

AEArrayData Field - kAEPackedArray[1]

kAEPackedArray[1] (char)

Array items consist of data of the same size and same type, and are packed without regard for word boundaries.

AEArrayData Field - kAEHandleArray[1]

kAEHandleArray[1] ([Handle](#))

Array items consist of handles to data of variable size and the same type.

AEArrayData Field - kAEDescArray[1]

kAEDescArray[1] ([AEDesc](#))

Array items consist of descriptor records of different descriptor types with data of variable size.

AEArrayData Field - kAEKeyDescArray[1]

kAEKeyDescArray[1] ([AEKeyDesc](#))

Array items consist of keyword-specified descriptor records with different keywords, different descriptor types, and data of variable size.

AEArrayDataPointer

A pointer a structure containing OSA event array data.

```
typedef AEArrayData *AEArrayDataPointer;
```

AEArrayType

Type of an OSA event array.

```
typedef unsigned char AEArrayType;
```

This data type can be set to any of the constants in the following list:

kAEDataArray	Array items consist of data of the same size and same type, and are aligned on word boundaries.
kAEDescArray	Array items consist of descriptor records of different descriptor types with data of variable size.
kAEHandleArray	Array items consist of handles to data of variable size and the same type.
kAEKeyDescArray	Array items consist of keyword-specified descriptor records with different keywords, different descriptor types, and data of variable size.
kAEPackedArray	Array items consist of data of the same size and same type, and are packed without regard for word boundaries.

AECOercionHandlerUPP

A pointer to a coercion handler routine.

```
typedef AECoeercePtrUPP AECoeercionHandlerUPP;
```

AECoeercePtrUPP

A pointer to a function.

```
typedef UniversalProcPtr AECoeercePtrUPP;
```

AEDesc

Descriptor record structure.

```
typedef struct _AEDesc {  
    DescType    descriptorType;  
    Handle      dataHandle;  
} AEDesc;
```

In OS/2, this structure must not be accessed directly by the programmer. The [AEClearDesc](#), [AEGetDescData](#), and [AESizeOfDescData](#) functions can be used to access the fields of an OS/2 AEDesc structure. The [AECreatDesc](#) function is used to create a new descriptor and initialize it with a type and data.

AEDesc Field - descriptorType

descriptorType ([DescType](#))

A four-character string of type [DescType](#) that indicates the type of data being passed.

AEDesc Field - dataHandle

dataHandle ([Handle](#))

A handle to the data being passed.

AEDescList

List of descriptor records.

```
typedef AEDesc AEDescList;
```

AEEventClass

Event class for a high-level event. See *OSA Event Registry: Standard Suites* for more information about event classes.

```
typedef unsigned long AEEventClass;
```

AEEventHandlerUPP

Pointer to an OSA event handler.

```
typedef AEEventHandlerProcPtr AEEventHandlerUPP;
```

AEEventID

An OSA event ID, for example, kAEGetData, the Get Data event.

```
typedef unsigned long AEEventID;
```

AEEventSource

The source of an OSA event.

```
typedef unsigned char AEEventSource;
```

This data type can be set to any of the constants in the following list:

Constant	Meaning
kAEUnknownSource	Source of OSA event is unknown.
kAEDirectCall	A direct call that bypassed the PM message queue.
kAESameProcess	Target application is also the source application.
kAELocalProcess	Source application is another process on the same computer as the target application.

AEInteractAllowed

Processes that can interact with the user.

```
typedef unsigned char AEInteractAllowed;
```

This data type can be set to any of the constants in the following list:

kAEInteractWithSelf

This flag allows the server application to interact with the user in response to an OSA event only when the client application and server application are the same—that is, only when the application is sending the OSA event to itself.

kAEInteractWithLocal

This flag allows the server application to interact with the user in response to an OSA event only if the client application is on the same computer as the server application; this is the default if the [AESetInteractionAllowed](#) function is not used.

kAEInteractWithAll

This flag allows the server application to interact with the user in response to an OSA event sent from any client application on any computer.

AEKeyDesc

Keyword-specific descriptor record structure.

```
typedef struct _AEKeyDesc {
    AEKeyword    descKey;
    AEDesc       descContent;
} AEKeyDesc;
```

AEKeyDesc Field - descKey

descKey ([AEKeyword](#))

A four-character code of type AEKeyword that identifies the data in the *descContent* field.

AEKeyDesc Field - descContent

descContent ([AEDesc](#))

A descriptor record of type [AEDesc](#).

AEKeyword

A constant value that indicates a particular type of descriptor record. See *OSA Event Registry: Standard Suites* for more information about keywords.

```
typedef unsigned long AEKeyword;
```

AERecord

List of keyword-specific descriptor records.

```
typedef AEDescList AERecord;
```

AESendMode

Flags that determine how an OSA event is sent.

```
typedef long AESendMode;
```

These mode flags are divided into five categories. The `kAEAlwaysInteract`, `kAECanInteract`, and `kAENeverInteract` flags are used to communicate the user interaction preference to the server application, and only one of these can be set. The `kAECanSwitchLayers` flag is used to set the application switch mode. The flags `kAENoReply`, `kAEQueueReply`, and `kAEWaitReply` specify the reply mode for an OSA event, and only one of these can be set. The last flag, `kAEWantReceipt`, sets the return receipt mode.

`kAEAlwaysInteract`

The server application can interact with the user in response to the OSA event-by convention, whenever the server application normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user. If this flag is set and the server allows interaction, `AEInteractWithUser` either brings the server application to the foreground or posts a notification request.

`kAECanInteract`

The server application can interact with the user in response to the OSA event-by convention, if the user needs to supply information to the server application to the foreground or posts a notification request. This flag is the default when an OSA event is sent to a local application.

`kAECanSwitchLayers`

If both the client and server allow interaction and if the client application is the active application on the local computer and is waiting for a reply (that is, it has set the `kAEWaitReply` flag), `AEInteractWithUser` brings the server directly to the foreground. Otherwise, `AEInteractWithUser` uses the NotificationManager to request that the user bring the server application to the foreground.

`kAENeverInteract`

The server application should never interact with the user in response to the OSA event. If this flag is set, `AEInteractWithUser` returns the `errAENoUserInteraction` result code. This flag is the default when an OSA event is sent to a remote application.

`kAENoReply`

Your application does not want a reply OSA event; the server processes your OSA event as soon as it has the opportunity.

`kAEQueueReply`

Your application wants a reply OSA event; the reply appears in your event queue as soon as the server has the opportunity to process and respond to your OSA event.

`kAEWaitReply`

Your application wants a reply OSA event and is willing to give up the processor while waiting for the reply; for example, if the server application is on the same computer as your application, your application yields the process to allow the server to respond to your OSA event. If you specify `kAEWaitReply`, you should provide an idle function.

The process will continue to receive all messages (events) on its message queue.

`kAEWantReceipt`

The sender wants to receive a return receipt for this OSA event from the Event Manager. (A return receipt means only that the receiving application accepted the OSA event; the OSA event may or may not be handled successfully after it is accepted.) If the receiving application does not send a returned receipt before the request times out, `AESend` returns `errAETimeout` as its function result.

AESendPriority

Send priority of an OSA event.

```
typedef short AESendPriority;
```

This parameter can be set to one of the following values:

kAENormalPriority

The OSA event is posted at the end of the event queue.

kAEHighPriority

The OSA event is posted at the front of the event queue.

AESystemHandler

System handler.

```
typedef struct _AESystemHandler {  
    char      *moduleName;  
    char      *procedureName;  
} AESystemHandler;
```

```
typedef AESystemHandler *pAESystemHandler;
```

The specified module and procedure names must be usable in the DosLoadModule and DosGetProcAddress functions.

AESystemHandler Field - moduleName

moduleName (char *)

Pointer to the DLL name.

AESystemHandler Field - procedureName

procedureName (char *)

Pointer to the procedure entry-point name.

APIRET

Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long APIRET;
```

BOOL

Boolean.

Valid values are:

- FALSE, which is 0
- TRUE, which is 1

```
typedef unsigned long BOOL;
```

CMGRString

A character buffer of 256 bytes used in Component Manager methods.

```
typedef char CMGRString[256];
```

ComponentDescription

Component description structure.

```
typedef struct _ComponentDescription {  
    OSType      componentType;  
    OSType      componentSubType;  
    OSType      componentManufacturer;  
    ODULong     componentFlags;  
    ODULong     componentFlagsMask;  
} ComponentDescription;
```

ComponentDescription Field - componentType

componentType (OSType)

The four-character type of the component, for example, kOSAComponentType (**osa**).

ComponentDescription Field - componentSubType

componentSubType (OSType)

The four-character subtype of the component, for example, kOSAGenericScriptingComponentSubtype (**gsc**).

ComponentDescription Field - componentManufacturer

componentManufacturer (OSType)

The four-character manufacturer of the component, for example, "IBM".

ComponentDescription Field - componentFlags

componentFlags (ODULong)

A flag used to query the Component Manager to find a scripting component with a particular capability or determine whether a particular scripting component supports a particular capability.

kOSASupportsCompiling

The scripting component supports routines for compiling scripts.

kOSASupportsGetSource

The scripting component supports routines for retrieving source data.

kOSASupportsAEC coercion

The scripting component supports routines for coercing data.

kOSASupportsAESending

The scripting component supports routines for creating and sending OSA events during script execution.

kOSASupportsRecording

The scripting component supports routines for recording OSA events.

kOSASupportsConvenience

The scripting component supports routines which execute scripts in a single step.

kOSASupportsDialects

The scripting component supports routines for handling dialects.

kOSASupportsEventHandling

The scripting component supports routines for event handling through script contexts.

ComponentDescription Field - componentFlagsMask

componentFlagsMask (ODULong)

A mask which defines which component flags are to be used in the search.

ComponentRegistryData

Component registry data structure.

```
typedef struct _ComponentRegistryData {  
    OSType      componentType;  
    OSType      componentSubType;  
    OSType      componentManufacturer;  
    ODULong     componentFlags;  
    ODULong     componentVersion;  
    CMGRString  componentDLL;  
} ComponentRegistryData;
```

ComponentRegistryData Field - componentType

componentType (OSType)

The four-character type of the component, for example, kOSAComponentType (osa).

ComponentRegistryData Field - componentSubType

componentSubType (OSType)

The four-character subtype of the component, for example, kOSAGenericScriptingComponentSubtype (**gsc**).

ComponentRegistryData Field - componentManufacturer

componentManufacturer (OSType)

The four-character manufacturer of the component, for example, "IBM".

ComponentRegistryData Field - componentFlags

componentFlags (ODULong)

A flag used to query the Component Manager to find a scripting component with a particular capability or determine whether a particular scripting component supports a particular capability.

kOSASupportsCompiling

The scripting component supports routines for compiling scripts.

kOSASupportsGetSource

The scripting component supports routines for retrieving source data.

kOSASupportsAECOercion

The scripting component supports routines for coercing data.

kOSASupportsAESending

The scripting component supports routines for creating and sending OSA events during script execution.

kOSASupportsRecording

The scripting component supports routines for recording OSA events.

kOSASupportsConvenience

The scripting component supports routines which execute scripts in a single step.

kOSASupportsDialects

The scripting component supports routines for handling dialects.

kOSASupportsEventHandling

The scripting component supports routines for event handling through script contexts.

ComponentRegistryData Field - componentVersion

componentVersion (ODULong)

The version number of the component to be registered.

ComponentRegistryData Field - componentDLL

componentDLL (CMGRString)

A string containing the DLL name.

DescPtr

A pointer to a descriptor record.

```
typedef AEDesc *DescPtr;
```

DescType

Descriptor type.

```
typedef unsigned long DescType;
```

Descriptor types represent various data types. See *OSA Event Registry: Standard Suites* for more information about descriptor types.

Environment

SOM environment information.

```
typedef struct _Environment {  
    exception_type    _major;  
    struct exception {  
        char          *_exception_name;  
        void          *_params;  
    } exception;  
    char              *_smdAnchor;  
} Environment;
```

This structure contains environmental information that can be passed between a caller and a called object when a method is executed. For example, it can be used to pass information about the user ID of a client or to return exception data to the client following a method call.

To set an exception value in the caller's Environment, a method implementation makes a call to the `somSetException` function; for example:

```
void somSetException ( Environment *env  
                      exception_type major,  
                      string exception_name,  
                      void *params);
```

where `env` is a pointer to the Environment passed to the method, `_major` is an exception type, `_exception_name` is the exception name, and `_params` is a pointer to an initialized exception allocated by calling `SOMMalloc`.

Note: The `somSetException` method simply sets the exception value; it does not perform exit processing. If there are multiple calls to `somSetException` before the method returns, the call sees only the last exception value.

After a method returns, the calling client program can look at the Environment structure to determine whether an exception occurred. If `_major` does not equal `NO_EXCEPTION`, an exception was returned by the call. The user can retrieve the exception name and values by calling `somExceptionID` to retrieve a string containing the exception name and `somExceptionValue` to retrieve a pointer to the value of the exception. If `NULL` is passed as the Environment pointer in either of these calls, an implicit call is made to `somGetGlobalEnvironment`.

Environment Field - _major

_major ([exception_type](#))

Type of exception returned from the call.

Possible values are described in the following list:

NO_EXCEPTION

An exception was not returned.

USER_EXCEPTION

A user exception was returned.

SYSTEM_EXCEPTION

A system exception was returned.

Environment Field - __exception_name

__exception_name (char *)

String containing the exception name.

Environment Field - __params

__params

Pointer to an initialized exception structure that must be allocated by SOMMalloc.

Environment Field - __somedAnchor

__somedAnchor (char *)

exception_type

An exception type.

```
enum exception_type {  
    NO_EXCEPTION;  
    USER_EXCEPTION;  
    SYSTEM_EXCEPTION;  
};
```

exception_type Field - NO_EXCEPTION

NO_EXCEPTION
An exception was not returned.

exception_type Field - USER_EXCEPTION

USER_EXCEPTION
A user exception was returned.

exception_type Field - SYSTEM_EXCEPTION

SYSTEM_EXCEPTION
A system exception was returned.

Handle

Address of a pointer to a char.

```
typedef Ptr *Handle;
```

LONG

Signed integer in the range -2 147 483 648 through 2 147 483 647.

```
#define LONG long
```

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to -134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to -32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range -32 768 through 32 767.

ODBoolean

A Boolean value; the size of this type is platform-dependent.

```
typedef boolean ODBoolean;
```

```
kODTrue      True
kODFalse     False
```

ODSLong

A signed, 32-bit integer value.

```
typedef long ODSLong;
```

ODSShort

A 16-bit signed value.

```
typedef short ODSShort;
```

ODUByte

An unsigned 8-bit value.

```
typedef octet ODUByte;
```

ODULong

An unsigned 32-bit value.

```
typedef unsigned long ODULong;
```

OSAActiveUPP

A pointer to the active function.

```
typedef void *OSAActiveUPP;
```

OSAEError

Type of result codes.

```
typedef OSErr OSAError;
```

See sections [Return Codes by Number](#) and [Return Codes by Name](#) for a complete listing of error codes.

OSAEvent

List of attributes and parameters necessary for an OSA event.

```
typedef AERecord OSAEvent;
```

OSAEventHandlerUPP

A pointer to an event handler.

```
typedef void *OSAEventHandlerUPP;
```

OSAID

Script ID.

```
typedef unsigned long OSAID;
```

OSASendUPP

A pointer to a send function.

```
typedef void *OSASendUPP;
```

OSErr

An error returned by the OSA Event Manager functions.

```
typedef LONG OSErr;
```

OSLAccessorUPP

A pointer to an object accessor function.

```
typedef OSLAccessorProcPtr OSLAccessorUPP;
```

OSLAdjustMarksUPP

A pointer to adjust marks function.

```
typedef OSLAdjustMarksProcPtr OSLAdjustMarksUPP;
```

OSLCompareUPP

A pointer to a compare function.

```
typedef OSLCompareProcPtr OSLCompareUPP;
```

OSLCountUPP

A pointer to a count function.

```
typedef OSLCountProcPtr OSLCountUPP;
```

OSLDisposeTokenUPP

A pointer to a dispose token function.

```
typedef OSLDisposeTokenProcPtr OSLDisposeTokenUPP;
```

OSLGetErrDescUPP

A pointer to a get error description function.


```
typedef OSLGetErrDescProcPtr OSLGetErrDescUPP;
```

OSLGetMarkTokenUPP

A pointer to a get mark token function.

```
typedef OSLGetMarkTokenProcPtr OSLGetMarkTokenUPP;
```

OSLMarkUPP

A pointer to a mark function.

```
typedef OSLMarkProcPtr OSLMarkUPP;
```

OSType

A 32-bit unsigned integer.

```
typedef unsigned long OSType;
```

PID

Process identity.

```
typedef LHANDLE PID;
```

PCSZ

Pointer to a constant null-terminated string.

```
typedef const char *PCSZ;
```

ProcPtr

A pointer to a function.

```
typedef OSErr (* APIENTRY ProcPtr)();
```

PSZ

Pointer to a null-terminated string.

If you are using C++ **, you may need to use PCSZ.

```
typedef unsigned char *PSZ;
```

Ptr

Pointer to a char.

```
typedef char *Ptr;
```

SemanticEvent

A structure containing the OSA event data which is passed into the [AEProcessOSAEvent](#) function.

```
typedef struct _SemanticEvent {  
    ULONG      eventClass;  
    ULONG      eventID;  
    VOID       *eventData;  
} SemanticEvent;  
  
typedef SemanticEvent *pSemanticEvent;
```

SemanticEvent Field - eventClass

eventClass ([ULONG](#))
The class of the event.

SemanticEvent Field - eventID

eventID ([ULONG](#))

The event ID.

SemanticEvent Field - eventData

eventData

Event specific data.

Size

The size of a buffer.

```
typedef LONG Size;
```

SIZEL

Size structure (LONG values).

```
typedef struct _SIZEL {  
    LONG    cx; /* Width. */  
    LONG    cy; /* Height. */  
} SIZEL;  
  
typedef SIZEL *PSIZEL;
```

SIZEL Field - cx

cx (LONG)
Width.

SIZEL Field - cy

cy (LONG)
Height.

ScriptingComponentSelector

Scripting component selector.

```
typedef unsigned long ScriptingComponentSelector;
```

ULONG

32-bit unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

USHORT

Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

UniversalProcPtr

A pointer to a function.

```
typedef ProcPtr UniversalProcPtr;
```

VOID

A data area of undefined format.

```
#define VOID void
```

Constants

The following constants are used in the Open Scripting Architecture APIs.

Values Returned from the GetPageState Function

```

#define kPageInMemory      0
#define kPageOnDisk        1
#define kNotPaged          2
#define ErrOutOfMemory    -1
#define ErrMemoryProblem  -2

#define kDefaultHeapID      0

#define THEHEAPSIZE        8388600
#define SHRHEAPSIZE        4194300

```

Type Descriptors

```

#define typeBoolean        0x6C6F6F62 /* "loob" */
#define typeChar           0x54584554 /* "TXET" */
#define typeSMInt          0x726f6873 /* "shor" */
#define typeInteger        0x676E6F6C /* "long" */
#define typeSMFloat        0x676E6973 /* "gnis" */
#define typeFloat          0x62756F64 /* "buod" */
#define typeLongInteger    0x676E6F6C /* "long" */
#define typeShortInteger   0x726F6873 /* "shor" */
#define typeLongFloat      0x62756F64 /* "buod" */
#define typeShortFloat     0x676E6973 /* "gnis" */
#define typeExtended       0x65747865 /* "etxe" */
#define typeComp           0x706d6F63 /* "pmoc" */
#define typeMagnitude      0x6E67616D /* "magn" */
#define typeAEList         0x7473696C /* "tsil" */
#define typeAERecord       0x6F636572 /* "ocer" */
#define typeOSAEvent       0x74766561 /* "tvea" */
#define typeTrue           0x65757274 /* "eurt" */
#define typeFalse          0x736C6166 /* "slaf" */
#define typeAlias          0x73696C61 /* "sila" */
#define typeEnumerated     0x6D756E65 /* "mune" */
#define typeType           0x65707974 /* "epyt" */
#define typeAppParameters  0x61707061 /* "appa" */
#define typeProperty       0x706F7270 /* "porp" */
#define typeFSS            0x20737366 /* "ssf" */
#define typeKeyword        0x7779656B /* "wyek" */
#define typeSectionH       0x74636573 /* "tces" */
#define typeWildcard       0x2A2A2A2A /* "****" */
#define typeApplSignature  0x6E676973 /* "ngis" */
#define typeSessionID      0x64697373 /* "diss" */
#define typeTargetID       0x67726174 /* "grat" */
#define typeProcessSerialNumber 0x206E7370 /* "nsp" */
#define typeNull           0x6C6C756E /* "llun" */
#define typeLELongInteger  0x676E6F6C /* "gnol" */

```

```
#define typeLEShortInteger      0x726F6873 /* "rohs" */
#define typeLEMagnitude        0x6E67616D /* "ngam" */
```

New Address Descriptors for OS/2

```
#define typePID                0x20444950 /* "DIP" */
#define typeTCPIP              0x49504354 /* "IPCT" */
```

New Little Endian Desc Types for OS2

```
#define typeLELongInteger      0x676E6F6C /* "gnol" */
#define typeLEShortInteger     0x726F6873 /* "rohs" */
#define typeLEMagnitude        0x6E67616D /* "ngam" */
```

Keywords for OSA Event Parameters

```
#define keyDirectObject        0x2D2D2D2D /* "----" */
#define keyErrorNumber         0x6E727265 /* "nrre" */
#define keyErrorString         0x73727265 /* "srre" */
#define keyProcessSerialNumber 0x206E7370 /* "nsp" */
```

Keywords for OSA Event Attributes

```
#define keyTransactionIDAttr    0x6E617274 /* "nart" */
#define keyReturnIDAttr        0x64697472 /* "ditr" */
#define keyEventClassAttr      0x6C637665 /* "lcve" */
#define keyEventIDAttr         0x64697665 /* "dive" */
#define keyAddressAttr         0x72646461 /* "rdda" */
```

```
#define keyOptionalKeywordAttr 0x6B74706F /* "ktpo" */
#define keyTimeoutAttr         0x6F6D6974 /* "omit" */
#define keyInteractLevelAttr   0x65746E69 /* "etni" */
#define keyEventSourceAttr     0x63727365 /* "crse" */
#define keyMissedKeywordAttr   0x7373696D /* "ssim" */
#define keyOriginalAddressAttr 0x6D6F7266 /* "morf" */
```

Keywords for Special Handlers

```
#define keyPreDispatch          0x63616870 /* "cahp" */
#define keySelectProc           0x686C6573 /* "hles" */
```

Keyword for Recording

```
#define keyAERecorderCount      0x72656372 /* "rcer" */
```

Keyword for Version Information

```
#define keyAEVersion            0x76657273 /* "srev" */
```

Event Class

```
#define kCoreEventClass         0x74766561 /* "tvea" */
```

Event IDs

```

#define kAEOpenApplication      0x7070616F /* "ppao" */
#define kAEOpenDocuments      0x636F646F /* "codo" */
#define kAEPrintDocuments      0x636F6470 /* "codp" */
#define kAEQuitApplication     0x74697571 /* "tiuq" */
#define kAEAnswer              0x72736E61 /* "rsna" */
#define kAEApplicationDied     0x7469626F /* "tibo" */

```

Constants for Use in AESend Mode

```

#define kAENoReply              0x00000001 /*sender doesn't want a */
                                   /* reply to event*/
#define kAEQueueReply          0x00000002 /*sender wants a reply but */
                                   /* won't wait*/
#define kAEWaitReply           0x00000003 /*sender wants a reply and */
                                   /* will wait*/
#define kAENeverInteract       0x00000010 /*server should not interact */
                                   /* with user*/
#define kAECanInteract         0x00000020 /*server may try to interact */
                                   /* with user*/
#define kAEAlwaysInteract      0x00000030 /*server should always */
                                   /* interact with user where */
                                   /* appropriate*/
#define kAECanSwitchLayer      0x00000040 /*interaction may switch layer*/
#define kAEDontReconnect       0x00000080 /*don't reconnect if there */
                                   /* is a sessClosedErr from */
                                   /* PPCToolbox*/
#define kAEWantReceipt         0x00000200 /*sender wants a receipt of */
                                   /* message */
#define kAEDontRecord          0x00001000 /*don't record this event - */
                                   /* available only in vers */
                                   /* 1.0.1 and greater*/
#define kAEDontExecute         0x00002000 /*record but don't send the */
                                   /* event - available only in */
                                   /* vers 1.0.1 and greater*/

```

Constants for the Send Priority in AESend

```

#define kAENormalPriority       0x00000000 /*post message at the end */

```



```

#define kAEHighPriority          0x00000001    /* of the event queue*/
/*post message at the front */
/* of the event queue*/

```

Constants for Recording

```

#define kAESTartRecording        0x72656361 /* "acer" */
#define kAESTopRecording         0x72656363 /* "ccer" */
#define kAENotifyStartRecording  0x72656331 /* "lcer" */
#define kAENotifyStopRecording    0x72656330 /* "0cer" */
#define kAENotifyRecording        0x72656372 /* "rcer" */

```

Constant for the returnID Parameter of AECreatEOSAEvent

```

#define kAutoGenerateReturnID    -1            /*AECreatEOSAEvent will */
/*generate a session-unique I

```

Constant for Transaction IDs

```

#define kAnyTransactionID        0            /*no transaction is in use*/

```

Constants for Timeout Durations

```

#define kAEDefaultTimeout        -1            /*timeout value determined */
/* by AEM*/
#define kNoTimeOut               -2            /*wait until reply comes */
/* back, however long it take

```

Constants for AEResumeTheCurrentEvent

```
#define kAENoDispatch          0          /*dispatch parameter to */
                                   /* AEResumeTheCurrentEvent */
                                   /* takes a pointer to a dispa
#define kAEUseStandardDispatch 0xFFFFFFFF /*table, or one of these */
                                   /* two constants*/
```

Constants for Reserved Process IDs

```
#define kCurrentProcess        0L
```

Constants for AESendPriority in AESend

```
enum {
    kAEInteractWithSelf,
    kAEInteractWithLocal,
    kAEInteractWithAll
};
```

Constants for AEInteractAllowed

```
enum {
    kAEUnknownSource,
    kAEDirectCall,
    kAESameProcess,
    kAELocalProcess,
    kAERemoteProcess
```

```
};
```

Constants for AEEventSource

```
enum {
    kAEDataArray,
    kAEPackedArray,
    kAEHandleArray,
    kAEDescArray,
    kAEKeyDescArray
};
```

Object and Element Classes

#define CAELIST	0x7473696C	/* "tsil" */
#define CAPPLICATION	0x70706163	/* "ppac" */
#define CARC	0x63726163	/* "crac" */
#define CBOOLEAN	0x6C6F6F62	/* "loob" */
#define CCELL	0x6C656363	/* "lecc" */
#define CCHAR	0x20616863	/* "ahc" */
#define CCOLORTABLE	0x74726C63	/* "trlc" */
#define CCOLUMN	0x6C6F6363	/* "locc" */
#define CDOCUMENT	0x75636F64	/* "ucod" */
#define CDRAWINGAREA	0x77726463	/* "wrdc" */
#define CENUMERATION	0x6D756E65	/* "mune" */
#define CFILE	0x656C6966	/* "elif" */
#define CFIXED	0x64786966	/* "dxif" */
#define CFIXEDPOINT	0x746E7066	/* "tnpf" */
#define CFIXEDRECTANGLE	0x74637266	/* "tcrf" */
#define CGRAPHICLINE	0x6E696C67	/* "nilg" */
#define CGRAPHICOBJECT	0x626F6763	/* "bogc" */
#define CGRAPHICSHAPE	0x68736763	/* "hsgc" */
#define CGRAPHICTEXT	0x78746763	/* "xtgc" */
#define CGROUPEDGRAPHIC	0x63697063	/* "cipc" */
#define CINSERTIONLOC	0x6C736E69	/* "lsni" */
#define CINSERTIONPOINT	0x736E6963	/* "snic" */
#define CINTLTEXT	0x74787469	/* "txti" */
#define CINTLWRITINGCODE	0x6C746E69	/* "ltni" */
#define CITEM	0x6D746963	/* "mtic" */
#define CLINE	0x6E696C63	/* "nilc" */
#define CLONGDATETIME	0x2074646C	/* "tdl" */

```

#define CLONGFIXED          0x6478666C    /* "dxfl" */
#define CLONGFIXEDPOINT    0x7470666C    /* "tpfl" */
#define CLONGFIXEDRECTANGLE 0x6372666C    /* "crfl" */
#define CLONGINTEGER       0x676E6F6C    /* "gnol" */
#define CLONGPOINT         0x746E706C    /* "tnpl" */
#define CLONGRECTANGLE     0x7463726C    /* "tcrl" */
#define CMACHINELOC        0x636F4C6D    /* "coLm" */
#define CMENU              0x756E6D63    /* "unmc" */
#define CMENUITEM          0x6E656D63    /* "nemc" */
#define COBJECT            0x6A626F63    /* "jboc" */
#define COBJECTSPECIFIER   0x206A626F    /* " jbo" */
#define COPENABLEOBJECT    0x626F6F63    /* "booc" */
#define COVAL              0x6C766F63    /* "lvoc" */
#define CPARAGRAPH         0x72617063    /* "rapc" */
#define CPICT              0x54434950    /* "TCIP" */
#define CPIXEL             0x6C787063    /* "lxpc" */
#define CPIXELMAP          0x78697063    /* "xipc" */
#define CPOLYGON           0x6E677063    /* "ngpc" */
#define CPROPERTY          0x706F7270    /* "porp" */
#define CQDPOINT           0x74704451    /* "tpDQ" */
#define CQDRECTANGLE       0x74726471    /* "trdq" */
#define CRECTANGLE         0x63657263    /* "cerc" */
#define CRGBCOLOR          0x42475263    /* "BGRc" */
#define CROTATION          0x746F7274    /* "tort" */
#define CROUNDEDRECTANGLE  0x63727263    /* "crrc" */
#define CROW               0x776F7263    /* "worc" */
#define CSELECTION          0x6C657363    /* "lesc" */
#define CSHORTINTEGER      0x726F6873    /* "rohs" */
#define CTABLE             0x6C627463    /* "lbtc" */
#define CTEXT              0x74787463    /* "txtc" */
#define CTEXTFLOW          0x6F6C6663    /* "olfc" */
#define CTEXTSTYLES        0x79747374    /* "ytst" */
#define CTYPE              0x65707974    /* "epyt" */
#define CVERSION           0x73726576    /* "srev" */
#define CWINDOW            0x6E697763    /* "niwc" */
#define CWORD              0x726F7763    /* "rowc" */

```

```

#define cAEList            CAELIST
#define cApplication       CAPPLICATION
#define cArc               CARC
#define cBoolean           CBOOLEAN
#define cCell              CCELL
#define cChar              CCHAR
#define cColorTable        CCOLORTABLE
#define cColumn            CCOLUMN
#define cDocument          CDOCUMENT
#define cDrawingArea       CDRAWINGAREA
#define cEnumeration       CENUMERATION
#define cFile              CFILE
#define cFixed             CFIXED
#define cFixedPoint        CFIXEDPOINT
#define cFixedRectangle    CFIXEDRECTANGLE
#define cGraphicLine       CGRAPHICLINE
#define cGraphicObject     CGRAPHICOBJECT
#define cGraphicShape      CGRAPHICSHAPE

```

#define cGraphicText	CGRAPHICTEXT	
#define cGroupedGraphic	CGROUPEDGRAPHIC	
#define cInsertionLoc	CINSERTIONLOC	
#define cInsertionPoint	CINSERTIONPOINT	
#define cIntlText	CINTLTEXT	
#define cIntlWritingCode	CINTLWRITINGCODE	
#define cItem	CITEM	
#define cLine	CLINE	
#define cLongDateTime	CLONGDATETIME	
#define cLongFixed	CLONGFIXED	
#define cLongFixedPoint	CLONGFIXEDPOINT	
#define cLongFixedRectangle	CLONGFIXEDRECTANGLE	
#define cLongInteger	CLONGINTEGER	
#define cLongPoint	CLONGPOINT	
#define cLongRectangle	CLONGRECTANGLE	
#define cMachineLoc	CMACHINELOC	
#define cMenu	CMENU	
#define cMenuItem	CMENUITEM	
#define cObject	COBJECT	
#define cObjectSpecifier	COBJECTSPECIFIER	
#define cOpenableObject	COPENABLEOBJECT	
#define cOval	COVAL	
#define cParagraph	CPARAGRAPH	
#define cPICT	CPICT	
#define cPixel	CPIXEL	
#define cPixelMap	CPIXELMAP	
#define cPolygon	CPOLYGON	
#define cProperty	CPROPERTY	
#define cQDPoint	CQDPOINT	
#define cQDRectangle	CQDRECTANGLE	
#define cRectangle	CRECTANGLE	
#define cRGBColor	CRGBCOLOR	
#define cRotation	CROTATION	
#define cRoundedRectangle	CROUNDEDRECTANGLE	
#define cRow	CROW	
#define cSelection	CSELECTION	
#define cShortInteger	CSHORTINTEGER	
#define cTable	CTABLE	
#define cText	CTEXT	
#define cTextFlow	CTEXTFLOW	
#define cTextStyles	CTEXTSTYLES	
#define cType	CTYPE	
#define cVersion	CVERSION	
#define cWindow	CWINDOW	
#define cWord	CWORD	
#define ENUMARROWS	0x6F727261	/* "orra" */
#define ENUMJUSTIFICATION	0x7473756A	/* "tsuj" */
#define ENUMKEYFORM	0x6D72666B	/* "mrfk" */
#define ENUMPOSITION	0x69736F70	/* "isop" */
#define ENUMPROTECTION	0x6E747270	/* "ntrp" */
#define ENUMQUALITY	0x6C617571	/* "lauq" */
#define ENUMSAVEOPTIONS	0x6F766173	/* "ovas" */
#define ENUMSTYLE	0x6C797473	/* "lyts" */
#define ENUMTRANSFERMODE	0x6E617274	/* "nart" */

#define enumJustification	ENUMJUSTIFICATION
#define enumKeyForm	ENUMKEYFORM
#define enumPosition	ENUMPOSITION
#define enumProtection	ENUMPROTECTION
#define enumQuality	ENUMQUALITY
#define enumSaveOptions	ENUMSAVEOPTIONS
#define enumStyle	ENUMSTYLE
#define enumTransferMode	ENUMTRANSFERMODE

Keywords

#define KAEABOUT	0x756F6261	/* "uoba" */
#define KAEAFTER	0x65746661	/* "etfa" */
#define KAEALIASSELECTION	0x696C6173	/* "ilas" */
#define KAEALLCAPS	0x70636C61	/* "pcla" */
#define KAEARROWATEND	0x6E657261	/* "nera" */
#define KAEARROWATSTART	0x74737261	/* "tsra" */
#define KAEARROWBOTHENDS	0x6F627261	/* "obra" */
#define KAEASK	0x206B7361	/* " ksa" */
#define KAEBEFORE	0x6F666562	/* "ofeb" */
#define KAEBEGINNING	0x676E6762	/* "gn gb" */
#define KAEBEGINSWITH	0x74776762	/* "tw gb" */
#define KAEBEGINTRANSACTION	0x69676562	/* "ig eb" */
#define KAEBOLD	0x646C6F62	/* "dlob" */
#define KAECASESENSEQUALS	0x71657363	/* "qesc" */
#define KAECENTERED	0x746E6563	/* "tnec" */
#define KAECHANGEVIEW	0x77656976	/* "weiv" */
#define KAECLONE	0x6E6F6C63	/* "nolc" */
#define KAECLOSE	0x736F6C63	/* "solc" */
#define KAECONDENSED	0x646E6F63	/* "dnoc" */
#define KAECONTAINS	0x746E6F63	/* "tnoc" */
#define KAECOPY	0x79706F63	/* "ypoc" */
#define KAECORESUIE	0x65726F63	/* "eroc" */
#define KAECOUNTELEMENTS	0x65746E63	/* "etnc" */
#define KAECREATEELEMENT	0x6C657263	/* "lerc" */
#define KAECREATEPUBLISHER	0x62757063	/* "bupc" */
#define KAECUT	0x20747563	/* " tuc" */
#define KAEDELETE	0x6F6C6564	/* "oled" */
#define KAEDOOBJECTSEXIST	0x78656F64	/* "xeod" */
#define KAEDOSCRIP	0x63736F64	/* "csod" */
#define KAEDRAG	0x67617264	/* "gard" */
#define KAEDUPLICATESELECTION	0x70756473	/* "puds" */
#define KAEEDITGRAPHIC	0x74696465	/* "tide" */
#define KAEEMPTYTRASH	0x74706D65	/* "tpme" */
#define KAEEND	0x20646E65	/* " dne" */
#define KAEENDSWITH	0x73646E65	/* "sdne" */
#define KAEENDTRANSACTION	0x74646E65	/* "tdne" */

```

#define KAEEQUALS 0x2020203D /* " =" */
#define KAEEXPANDED 0x70786570 /* "pxep" */
#define KAEFAST 0x74736166 /* "tsaf" */
#define KAEFINDEREVENTS 0x52444E46 /* "RDNF" */
#define KAEFORMULAPROTECT 0x6F727066 /* "orpf" */
#define KAEFULLYJUSTIFIED 0x6C6C7566 /* "lluf" */
#define KAEGETCLASSINFO 0x6A626F71 /* "jboq" */
#define KAEGETDATA 0x64746567 /* "dteg" */
#define KAEGETDATASIZE 0x7A697364 /* "zisd" */
#define KAEGETEVENTINFO 0x69657467 /* "ietg" */
#define KAEGETINFOSELECTION 0x666E6973 /* "fnis" */
#define KAEGETPRIVILEGESELECTION 0x76727073 /* "vrps" */
#define KAEGETSUITEINFO 0x69737467 /* "istg" */
#define KAEGREATERTHAN 0x2020203E /* " >" */
#define KAEGREATERTHANEQUALS 0x20203D3E /* " =>" */
#define KAEGROW 0x776F7267 /* "worg" */
#define KAEHIDDEN 0x6E646968 /* "ndih" */
#define KAEHIQUALITY 0x75716968 /* "uqih" */
#define KAEIMAGEGRAPHIC 0x72676D69 /* "rgmi" */
#define KAEISUNIFORM 0x6E757369 /* "nusi" */
#define KAEITALIC 0x6C617469 /* "lati" */
#define KAELEFTJUSTIFIED 0x7466656C /* "tfel" */
#define KAELESSTHAN 0x2020203C /* " <" */
#define KAELESSTHANEQUALS 0x20203D3C /* " =<" */
#define KAELOWERCASE 0x63776F6C /* "cwol" */
#define KAEMAKEOBJECTSVISIBLE 0x7369766D /* "sivm" */
#define KAEMISCSTANDARDS 0x6373696D /* "csim" */
#define KAEMODIFIABLE 0x66646F6D /* "fdom" */
#define KAEMOVE 0x65766F6D /* "evom" */
#define KAENO 0x20206F6E /* " on" */
#define KAENOARROW 0x6F6E7261 /* "onra" */
#define KAENONMODIFIABLE 0x646F6D6E /* "domn" */
#define KAEOPEN 0x636F646F /* "codo" */
#define KAEOPENSELECTION 0x65706F73 /* "epos" */
#define KAEOUTLINE 0x6C74756F /* "ltuo" */
#define KAEPAGESETUP 0x75736770 /* "usgp" */
#define KAEPASTE 0x74736170 /* "tsap" */
#define KAEPLAIN 0x6E616C70 /* "nalp" */
#define KAEPRINT 0x636F6470 /* "codp" */
#define KAEPRINTSELECTION 0x69727073 /* "irps" */
#define KAEPRINTWINDOW 0x6E697770 /* "niwp" */
#define KAEPUTAWAYSELECTION 0x74757073 /* "tups" */
#define KAEQDADDOVER 0x6F646461 /* "odda" */
#define KAEQDADDPIN 0x70646461 /* "pdda" */
#define KAEQDADMAX 0x786D6461 /* "xmda" */
#define KAEQDADMIN 0x6E6D6461 /* "nmda" */
#define KAEQDBIC 0x20636962 /* " cib" */
#define KAEQDBLEND 0x646E6C62 /* "dnlb" */
#define KAEQDCOPY 0x20797063 /* " ypc" */
#define KAEQDNOTBIC 0x6369626E /* "cibn" */
#define KAEQDNOTCOPY 0x7970636E /* "ypcn" */
#define KAEQDNOTOR 0x726F746E /* "rotn" */
#define KAEQDNOTXOR 0x726F786E /* "roxn" */
#define KAEQDOR 0x2020726F /* " ro" */
#define KAEQDSUBOVER 0x6F627573 /* "obus" */

```

```

#define KAEQDSUBPIN 0x70627573 /* "pbus" */
#define KAEQDSUPPLEMENTALSUITE 0x70736471 /* "psdq" */
#define KAEQDXOR 0x20726F78 /* " rox" */
#define KAEQUICKDRAWSUITE 0x77726471 /* "wr dq" */
#define KAEQUITALL 0x61697571 /* "ai uq" */
#define KAEREDO 0x6F646572 /* "oder" */
#define KAEREGULAR 0x6C676572 /* "lger" */
#define KAEREPPLACE 0x636C7072 /* "clpr" */
#define KAEREQUIRED SUITE 0x64716572 /* "d qer" */
#define KAERESTART 0x74736572 /* "tser" */
#define KAEREVEAL SELECTION 0x76657273 /* "vers" */
#define KAEREVERT 0x74727672 /* "trvr" */
#define KAERIGHTJUSTIFIED 0x74686772 /* "thgr" */
#define KAESAVE 0x65766173 /* "evas" */
#define KAEESELECT 0x74636C73 /* "tcls" */
#define KAESETDATA 0x64746573 /* "dtes" */
#define KAESETPOSITION 0x6E736F70 /* "nsop" */
#define KAESHADOW 0x64616873 /* "dahs" */
#define KAESHOWCLIPBOARD 0x6C636873 /* "lchs" */
#define KAESHUTDOWN 0x74756873 /* "tuhs" */
#define KAESLEEP 0x70656C73 /* "pels" */
#define KAESMALLCAPS 0x70636D73 /* "pcms" */
#define KAESPECIALCLASSPROPERTIES 0x21234063 /* "!#@c" */
#define KAESTRIKETHROUGH 0x6B727473 /* "krts" */
#define KAESUBSCRIPT 0x63736273 /* "csbs" */
#define KAESUPERScript 0x63737073 /* "csps" */
#define KAETABLESUITE 0x736C6274 /* "slbt" */
#define KAETEXTSUITE 0x54584554 /* "TXET" */
#define KAETRANSACTIONTERMINATED 0x6D727474 /* "mrtt" */
#define KAEUNDERLINE 0x6C646E75 /* "ldnu" */
#define KAEUNDO 0x6F646E75 /* "odnu" */
#define KAEWHOLEWORDEQUALS 0x71657777 /* "qeww" */
#define KAEYES 0x20736579 /* " sey" */
#define KAEZOOM 0x6D6F6F7A /* "mooz" */

```

```

#define kAEAfter KAEAFTER
#define kAEAliasSelection KAEALIASSELECTION
#define kAEAllCaps KAEALLCAPS
#define kAEArrowAtEnd KAEARROWATEND
#define kAEArrowAtStart KAEARROWATSTART
#define kAEArrowBothEnds KAEARROWBOTHENDS
#define kAEAsk KAEASK
#define kAEBefore KAEBEFORE
#define kAEBeginning KAEBEGINNING
#define kAEBeginsWith KAEBEGINSWITH
#define kAEBeginTransaction KAEBEGINTRANSACTION
#define kAEBold KAEBOLD
#define kAECASESenseEquals KAECASESENSEEQUALS
#define kAECentered KAECENTERED
#define kAEChangeView KAECHANGEVIEW
#define kAEClone KAECLONE
#define kAEClose KAECLOSE
#define kAECondensed KAECONDENSED
#define kAEContains KAECONTAINS
#define kAECopy KAECOPY

```


#define KAECoreSuite	KAECORESUIE
#define KAECountElements	KAECOUNTELEMENTS
#define KAECreateElement	KAECREATEELEMENT
#define KAECreatePublisher	KAECREATEPUBLISHER
#define KAECut	KAECUT
#define KAEDelete	KAEDELETE
#define KAEDoObjectsExist	KAEDOOBJECTSEXIST
#define KAEDoScript	KAEDOSCRIPT
#define KAEDrag	KAEDRAG
#define KAEDuplicateSelection	KAEDUPLICATESELECTION
#define KAEEditGraphic	KAEDITGRAPHIC
#define KAEEmptyTrash	KAEMPTYTRASH
#define KAEEnd	KAEND
#define KAEEndsWith	KAEENDSWITH
#define KAEEndTransaction	KAEENDTRANSACTION
#define KAEEquals	KAEEQUALS
#define KAEExpanded	KAEXPANDED
#define KAEFast	KAFAST
#define KAEFinderEvents	KAEFINDEREVENTS
#define KAEFormulaProtect	KAIFORMULAPROTECT
#define KAEFullyJustified	KAEFULLYJUSTIFIED
#define KAEGetClassInfo	KAEGETCLASSINFO
#define KAEGetData	KAEGETDATA
#define KAEGetDataSize	KAEGETDATASIZE
#define KAEGetEventInfo	KAEGETEVENTINFO
#define KAEGetInfoSelection	KAEGETINFOSELECTION
#define KAEGetPrivilegeSelection	KAEGETPRIVILEGESELECTION
#define KAEGetSuiteInfo	KAEGETSUITEINFO
#define KAEGreaterThan	KAEGREATERTHAN
#define KAEGreaterThanOrEquals	KAEGREATERTHANEQUALS
#define KAEGrow	KAEGROW
#define KAEHidden	KAEHIDDEN
#define KAEHiQuality	KAHIQUALITY
#define KAEImageGraphic	KAIMAGEGRAPHIC
#define KAEIsUniform	KAISUNIFORM
#define KAEItalic	KAETALIC
#define KAELeftJustified	KALEFTJUSTIFIED
#define KAELessThan	KAELESSTHAN
#define KAELessThanOrEquals	KAELESSTHANEQUALS
#define KAELowercase	KAELOWERCASE
#define KAEMakeObjectsVisible	KAEMAKEOBJECTSVISIBLE
#define KAEMiscStandards	KAEMISCSTANDARDS
#define KAEModifiable	KAEMODIFIABLE
#define KAEMove	KAEMOVE
#define KAENO	KAENO
#define KAENoArrow	KAENOWARROW
#define KAENonmodifiable	KAENONMODIFIABLE
#define KAEOpen	KAEOPEN
#define KAEOpenSelection	KAEOPENSELECTION
#define KAEOutline	KAEOUTLINE
#define KAEPageSetup	KAEPAGESETUP
#define KAEPaste	KAEPASTE
#define KAEPlain	KAEPLAIN
#define KAEPrint	KAEPRINT
#define KAEPrintSelection	KAEPRINTSELECTION

```

#define KAEPrintWindow          KAEPRINTWINDOW
#define KAEPutAwaySelection     KAEPUTAWAYSELECTION
#define KAEQDAddOver           KAEQDADDOVER
#define KAEQDAddPin            KAEQDADDPIN
#define KAEQDAdMax              KAEQDADMAX
#define KAEQDAdMin              KAEQDADMIN
#define KAEQDBic                KAEQDBIC
#define KAEQDBlend              KAEQDBLEND
#define KAEQDCopy               KAEQDCOPY
#define KAEQDNotBic             KAEQDNOTBIC
#define KAEQDNotCopy            KAEQDNOTCOPY
#define KAEQDNotOr              KAEQDNOTOR
#define KAEQDNotXor             KAEQDNOTXOR
#define KAEQDOr                 KAEQDOR
#define KAEQDSubOver            KAEQDSUBOVER
#define KAEQDSubPin             KAEQDSUBPIN
#define KAEQDSupplementalSuite  KAEQDSUPPLEMENTALSUITE
#define KAEQDXor                KAEQDXOR
#define KAEQuickdrawSuite       KAEQUICKDRAWSUITE
#define KAEQuitAll              KAEQUITALL
#define KAERedo                 KAEREDO
#define KAERegular              KAEREGULAR
#define KAEReplace              KAEREPLACE
#define KAERequiredSuite        KAEREQUIRED SUITE
#define KAERestart              KAERESTART
#define KAERevealSelection      KAEREVEALSELECTION
#define KAERever                KAEREVERT
#define KAERightJustified       KAERIGHTJUSTIFIED
#define KAESave                KAESAVE
#define KAESelect               KAESELECT
#define KAESetData              KAESETDATA
#define KAESetPosition          KAESETPOSITION
#define KAEShadow               KAESHADOW
#define KAEShowClipboard        KAESHOWCLIPBOARD
#define KAEShutDown             KAESHUTDOWN
#define KAESleep                KAESLEEP
#define KAESmallCaps            KAESMALLCAPS
#define KAESpecialClassProperties KAESPECIALCLASSPROPERTIES
#define KAESTrikethrough        KAESTRIKETHROUGH
#define KAESubscript            KAESUBSCRIPT
#define KAESuperscript          KAESUPERScript
#define KAETableSuite           KAETABLESUITE
#define KAETextSuite            KAETEXTSUITE
#define KAETransactionTerminated KAETRANSACTIONTERMINATED
#define KAEUnderline            KAEUNDERLINE
#define KAEUndo                 KAEUNDO
#define KAEWholeWordEquals      KAEWHOLEWORDEQUALS
#define KAEYes                  KAEYES
#define KAEZoom                 KAEZOOM

#define KEYAEANGLE              0x676E616B    /* "gnak" */
#define KEYAEARCANGLE           0x63726170    /* "crap" */
#define KEYAEBASEADDR           0x64646162    /* "ddab" */
#define KEYAEBESTTYPE           0x74736270    /* "tsbp" */
#define KEYAEBGNDCOLOR          0x6C636266    /* "lcbk" */

```

```

#define KEYAEBGNDPATTERN      0x7470626B    /* "tpbk" */
#define KEYAEBOUNDS           0x646E6270    /* "dnbp" */
#define KEYAECELLLIST         0x746C636B    /* "tlck" */
#define KEYAECLASSID          0x44496C63    /* "Dilc" */
#define KEYAECOLOR            0x726C6F63    /* "rloc" */
#define KEYAECOLORTABLE       0x62746C63    /* "btlc" */
#define KEYAECURVEHEIGHT      0x6468636B    /* "bhck" */
#define KEYAECURVEWIDTH       0x6477636B    /* "dwck" */
#define KEYAEDASHSTYLE        0x74736470    /* "tsdp" */
#define KEYAEADATA            0x61746164    /* "atad" */
#define KEYAEDEFAULTTYPE      0x74666564    /* "tfed" */
#define KEYAEDEFINITIONRECT   0x74726470    /* "trdp" */
#define KEYAEDESCTYPE         0x70747364    /* "ptsd" */
#define KEYAEDESTINATION      0x74736564    /* "tsed" */
#define KEYAEEDOANTIALIAS     0x61746E61    /* "atna" */
#define KEYAEDODITHERED      0x74696467    /* "tidg" */
#define KEYAEDOROTATE         0x7472646B    /* "trdk" */
#define KEYAEDOSCALE          0x6163736B    /* "acsk" */
#define KEYAEDOTTRANSLATE     0x6172746B    /* "artk" */
#define KEYAEEDITIONFILELOC   0x636F6C65    /* "cole" */
#define KEYAEELEMENTS         0x736D6C65    /* "smle" */
#define KEYAEENDPOINT         0x646E6570    /* "dnep" */
#define KEYAEEVENTCLASS       0x6C637665    /* "lcve" */
#define KEYAEEVENTID          0x69747665    /* "itve" */
#define KEYAEFILE             0x6C69666B    /* "lifk" */
#define KEYAEFILETYPE         0x70746C66    /* "ptlf" */
#define KEYAEFILLCOLOR        0x6C636C66    /* "lclf" */
#define KEYAEFILLPATTERN      0x74706C66    /* "tplf" */
#define KEYAEFLIPHORIZONTAL    0x6F68666B    /* "ohfk" */
#define KEYAEFLIPVERTICAL     0x7476666B    /* "tvfk" */
#define KEYAEFONT             0x746E6F66    /* "tnof" */
#define KEYAEFORMULA          0x726F6670    /* "rofp" */
#define KEYAEGRAPHICOBJECTS   0x73626F67    /* "sbog" */
#define KEYAEID               0x20204449    /* "DI" */
#define KEYAEIMAGEQUALITY     0x61757167    /* "auqg" */
#define KEYAEINSERTHERE       0x68736E69    /* "hsni" */
#define KEYAEKEYFORMS         0x6679656B    /* "fyek" */
#define KEYAEKEYWORD          0x6477796B    /* "dwyk" */
#define KEYAELEVEL            0x6C76656C    /* "lvel" */
#define KEYAELINEARROW        0x6F727261    /* "orra" */
#define KEYAENAME             0x6D616E70    /* "manp" */
#define KEYAENEWELEMENTLOC    0x6C656E70    /* "lenp" */
#define KEYAEOBJECT           0x6A626F6B    /* "jbok" */
#define KEYAEOBJECTCLASS      0x6C636F6B    /* "lcok" */
#define KEYAEOFFSTYLES        0x7473666F    /* "tsfo" */
#define KEYAEONSTYLES         0x74736E6F    /* "tsno" */
#define KEYAEPARAMETERS       0x736D7270    /* "smrp" */
#define KEYAEPARAMFLAGS       0x67666D70    /* "gfmp" */
#define KEYAEPENCOLOR         0x6C637070    /* "lcpp" */
#define KEYAEPENPATTERN       0x61707070    /* "appp" */
#define KEYAEPENWIDTH         0x64777070    /* "dwpp" */
#define KEYAEPixelDEPTH       0x74706470    /* "tpdp" */
#define KEYAEPixmapMINUS      0x6D6D706B    /* "mmpk" */
#define KEYAEPMTABLE          0x746D706B    /* "tmpk" */
#define KEYAEPPOINTLIST       0x746C7470    /* "tltp" */

```

```

#define KEYAEPOINTSIZE      0x7A737470    /* "zstp" */
#define KEYAEPOSITION      0x736F706B    /* "sopk" */
#define KEYAEPROPDATA      0x74647270    /* "tdrp" */
#define KEYAEPROPERTIES    0x6F727071    /* "orpq" */
#define KEYAEPROPERTY      0x7072706B    /* "prpk" */
#define KEYAEPROPFLAGS     0x67667270    /* "gfrp" */
#define KEYAEPROPID        0x706F7270    /* "porp" */
#define KEYAEPROTECTION    0x6F727070    /* "orpp" */
#define KEYAERENDERAS      0x6E65726B    /* "nerk" */
#define KEYAEREQUESTEDTYPE 0x70797472    /* "pytr" */
#define KEYAERESULT        0x2D2D2D2D    /* "----" */
#define KEYAERESULTINFO    0x6E697372    /* "nistr" */
#define KEYAEROTATION      0x746F7270    /* "torp" */
#define KEYAEROTPOINT      0x7074726B    /* "ptrk" */
#define KEYAEROWLIST       0x736C726B    /* "slrk" */
#define KEYAESAVEOPTIONS    0x6F766173    /* "ovas" */
#define KEYAESCALE         0x6C637370    /* "lcsp" */
#define KEYAESCRIPTTAG     0x74637370    /* "tcsp" */
#define KEYAESHOWWHERE     0x776F6873    /* "wohs" */
#define KEYAESTARTANGLE    0x676E6170    /* "gnap" */
#define KEYAESTARTPOINT    0x70747370    /* "ptsp" */
#define KEYAESTYLES        0x7974736B    /* "ytsk" */
#define KEYAESUITEID       0x74697573    /* "tius" */
#define KEYAETEXT          0x7478746B    /* "txtk" */
#define KEYAETEXTCOLOR     0x63787470    /* "cxtp" */
#define KEYAETEXTFONT      0x66787470    /* "fxtp" */
#define KEYAETEXTPOINTSIZE 0x73707470    /* "sptp" */
#define KEYAETEXTSTYLES    0x74737874    /* "tsxt" */
#define KEYAETHETEXT       0x78746874    /* "xtht" */
#define KEYAETRANSFERMODE   0x6D747070    /* "mtpp" */
#define KEYAETRANSLATION   0x73727470    /* "srtp" */
#define KEYAETRYASSTRUCTGRAF 0x676F6F74    /* "goot" */
#define KEYAEUNIFORMSTYLES 0x6C747375    /* "ltsu" */
#define KEYAEUPDATEON      0x64707570    /* "dpup" */
#define KEYAEUSERTERM      0x6D727475    /* "mrtu" */
#define KEYAEWINDOW        0x77646E77    /* "wdnw" */
#define KEYAEWRITINGCODE   0x64637277    /* "dcrw" */
#define KEYAETSMSCRIPTTAG  0x676C6373    /* "glcs" */
#define KEYAETSMTEXTFONT   0x6678746B    /* "fxtk" */
#define KEYAETSMTEXTPOINTSIZE 0x7370746B    /* "sptk" */
#define KEYMISCELLANEOUS   0x63736D66    /* "csmf" */
#define KEYSELECTION       0x6C657366    /* "lesf" */
#define KEYWINDOW          0x646E776B    /* "dnwk" */

#define keyAEAngle          KEYAEANGLE
#define keyAEArcAngle       KEYAEARCANGLE
#define keyAEBaseAddr       KEYAEBASEADDR
#define keyAEBestType       KEYAEBESTTYPE
#define keyAEBgndColor      KEYAEBGNDCOLOR
#define keyAEBgndPattern    KEYAEBGNDPATTERN
#define keyAEBounds         KEYAEBOUNDS
#define keyAECellList       KEYAECELLLIST
#define keyAEClassID        KEYAECLASSID
#define keyAECOLOR          KEYAECOLOR
#define keyAECOLORTable     KEYAECOLORTABLE

```

#define keyAECurveHeight	KEYAECURVEHEIGHT
#define keyAECurveWidth	KEYAECURVEWIDTH
#define keyAEDashStyle	KEYAEDASHSTYLE
#define keyAEData	KEYAEDATA
#define keyAEDefaultType	KEYAEDEFAULTTYPE
#define keyAEDefinitionRect	KEYAEDEFINITIONRECT
#define keyAEDescType	KEYAEDESCTYPE
#define keyAEDestination	KEYAEDESTINATION
#define keyAEDoAntiAlias	KEYAEDOANTIALIAS
#define keyAEDoDithered	KEYAEDODITHERED
#define keyAEDoRotate	KEYAEDOROTATE
#define keyAEDoScale	KEYAEDOSCALE
#define keyAEDoTranslate	KEYAEDOTRANSLATE
#define keyAEEditionFileLoc	KEYAEEDITIONFILELOC
#define keyAEElements	KEYAEELEMENTS
#define keyAEEndPoint	KEYAEENDPOINT
#define keyAEEventClass	KEYAEEVENTCLASS
#define keyAEEventID	KEYAEEVENTID
#define keyAEFile	KEYAEFILE
#define keyAEFileType	KEYAEFILETYPE
#define keyAEFillColor	KEYAEFILLCOLOR
#define keyAEFillPattern	KEYAEFILLPATTERN
#define keyAEFlipHorizontal	KEYAEFLIPHORIZONTAL
#define keyAEFlipVertical	KEYAEFLIPVERTICAL
#define keyAEFont	KEYAEFONT
#define keyAEFormula	KEYAEFORMULA
#define keyAEGraphicObjects	KEYAEGRAPHICOBJECTS
#define keyAEID	KEYAEID
#define keyAEImageQuality	KEYAEIMAGEQUALITY
#define keyAEInsertHere	KEYAEINSERTHERE
#define keyAEKeyForms	KEYAEKEYFORMS
#define keyAEKeyword	KEYAEKEYWORD
#define keyAELevel	KEYAELEVEL
#define keyAELineArrow	KEYAELINEARROW
#define keyAENAME	KEYAENAME
#define keyAENewElementLoc	KEYAENEWELEMENTLOC
#define keyAEObject	KEYAEOBJECT
#define keyAEObjectClass	KEYAEOBJECTCLASS
#define keyAEOffStyles	KEYAEOFFSTYLES
#define keyAEOnStyles	KEYAEONSTYLES
#define keyAEPARAMETERS	KEYAEPARAMETERS
#define keyAEPARAMFLAGS	KEYAEPARAMFLAGS
#define keyAEPENCOLOR	KEYAEPENCOLOR
#define keyAEPENPATTERN	KEYAEPENPATTERN
#define keyAEPENWIDTH	KEYAEPENWIDTH
#define keyAEPixelDepth	KEYAEPixelDEPTH
#define keyAEPixMapMinus	KEYAEPixMAPMINUS
#define keyAEPMTABLE	KEYAEPMTABLE
#define keyAEPPOINTLIST	KEYAEPPOINTLIST
#define keyAEPPOINTSIZE	KEYAEPPOINTSIZE
#define keyAEPPOSITION	KEYAEPPOSITION
#define keyAEPROPDATA	KEYAEPROPDATA
#define keyAEPROPERTIES	KEYAEPROPERTIES
#define keyAEPROPERTY	KEYAEPROPERTY
#define keyAEPROPFLAGS	KEYAEPROPFLAGS

#define keyAEPPropID	KEYAEPROPID
#define keyAEProtection	KEYAEPROTECTION
#define keyAERenderAs	KEYAERENDERAS
#define keyAERequestedType	KEYAEREQUESTEDTYPE
#define keyAEResult	KEYAERESULT
#define keyAEResultInfo	KEYAERESULTINFO
#define keyAERotation	KEYAEROTATION
#define keyAERotPoint	KEYAEROTPOINT
#define keyAERowList	KEYAEROWLIST
#define keyAESaveOptions	KEYAESAVEOPTIONS
#define keyAEScale	KEYAESCALE
#define keyAEScriptTag	KEYAESCRIPTTAG
#define keyAEShowWhere	KEYAESHOWWHERE
#define keyAESTartAngle	KEYAESTARTANGLE
#define keyAESTartPoint	KEYAESTARTPOINT
#define keyAESTyles	KEYAESTYLES
#define keyAESuiteID	KEYAESUITEID
#define keyAEText	KEYAETEXT
#define keyAETextColor	KEYAETEXTCOLOR
#define keyAETextFont	KEYAETEXTFONT
#define keyAETextPointSize	KEYAETEXTPOINTSIZE
#define keyAETextStyles	KEYAETEXTSTYLES
#define keyAETheText	KEYAETHETEXT
#define keyAETransferMode	KEYAETRANSFERMODE
#define keyAETranslation	KEYAETRANSLATION
#define keyAETryAsStructGraf	KEYAETRYASSTRUCTGRAF
#define keyAEUniformStyles	KEYAEUNIFORMSTYLES
#define keyAEUpdateOn	KEYAEUPDATEON
#define keyAEUserTerm	KEYAEUSERTERM
#define keyAEWindow	KEYAEWINDOW
#define keyAEWritingCode	KEYAEWRITINGCODE
#define keyAETSMScriptTag	KEYAETSMSCRIPTTAG
#define keyAETSMTextFont	KEYAETSMTEXTFONT
#define keyAETSMTextPointSize	KEYAETSMTEXTPOINTSIZE
#define keyMiscellaneous	KEYMISCELLANEOUS
#define keySelection	KEYSELECTION
#define keyWindow	KEYWINDOW

Properties

#define PARCANGLE	0x63726170	/* "crap" */
#define PBACKGROUNDColor	0x6C636270	/* "lcbp" */
#define PBACKGROUNDPattern	0x74706270	/* "tpbp" */
#define PBESTType	0x74736270	/* "tsbp" */
#define PBOUNDS	0x646E6270	/* "dnbp" */
#define PCLASS	0x736C6370	/* "slcp" */
#define PCLIPBOARD	0x696C6370	/* "ilcp" */
#define PCOLOR	0x726C6F63	/* "rloc" */

```

#define PCOLORTABLE 0x62746C63 /* "btlc" */
#define PCONTENTS 0x746E6370 /* "tncp" */
#define PCORNERCURVEHEIGHT 0x64686370 /* "dhcp" */
#define PCORNERCURVEWIDTH 0x64776370 /* "dwcp" */
#define PDASHSTYLE 0x74736470 /* "tsdp" */
#define PDEFAULTTYPE 0x74666564 /* "tfed" */
#define PDEFINITIONRECT 0x74726470 /* "trdp" */
#define PENABLED 0x6C626E65 /* "lbne" */
#define PENDPOINT 0x646E6570 /* "dnep" */
#define PFILLCOLOR 0x6C636C66 /* "lclf" */
#define PFILLPATTERN 0x74706C66 /* "tplf" */
#define PFONT 0x746E6F66 /* "tnof" */
#define PFORMULA 0x726F6670 /* "rofp" */
#define PGRAPHICOBJECTS 0x73626F67 /* "sbog" */
#define PHASCLOSEBOX 0x626C6368 /* "blch" */
#define PHASTITLEBAR 0x74697470 /* "titp" */
#define PAEID 0x20204449 /* " DI" */
#define PINDEX 0x78646970 /* "xdip" */
#define PINsertionLOC 0x736E6970 /* "snip" */
#define PISFLOATING 0x6C667369 /* "lfsi" */
#define PISFRONTPROCESS 0x66736970 /* "fsip" */
#define PISMODAL 0x646F6D70 /* "domp" */
#define PISMODIFIED 0x646F6D69 /* "domi" */
#define PISRESIZABLE 0x7A737270 /* "zsrp" */
#define PISSTATIONERYPAD 0x64707370 /* "dpsp" */
#define PISZOOMABLE 0x6D7A7369 /* "mzsi" */
#define PISZOOMED 0x6D757A70 /* "muzp" */
#define PITEMNUMBER 0x6E6D7469 /* "nmti" */
#define PJUSTIFICATION 0x74736A70 /* "tsjp" */
#define PLINEARROW 0x6F727261 /* "orra" */
#define PMENUID 0x64696E6D /* "dinm" */
#define PNAME 0x6D616E70 /* "manp" */
#define PNEWELEMENTLOC 0x6C656E70 /* "lnep" */
#define PPENCOLOR 0x6C637070 /* "lcpp" */
#define PPENPATTERN 0x61707070 /* "appp" */
#define PPENWIDTH 0x64777070 /* "dwpp" */
#define PPIXELDEPTH 0x74706470 /* "tpdp" */
#define PPOINTLIST 0x746C7470 /* "tltp" */
#define PPOINTSIZe 0x7A737470 /* "zstp" */
#define PPROTECTION 0x6F727070 /* "orpp" */
#define PROTATION 0x746F7270 /* "torp" */
#define PSCALE 0x6C637370 /* "lcsp" */
#define PSCRIPT 0x74706373 /* "tpcs" */
#define PSELECTED 0x636C6573 /* "cles" */
#define PSELECTION 0x656C6573 /* "eles" */
#define PSTARTANGLE 0x676E6170 /* "gnap" */
#define PSTARTPOINT 0x70747370 /* "ptsp" */
#define PTEXTCOLOR 0x63787470 /* "ctxp" */
#define PTEXTFONT 0x66787470 /* "ftxp" */
#define PTEXTITEMDELIMITERS 0x6C647874 /* "ldxt" */
#define PTEXTPOINTSIZE 0x73707470 /* "sptp" */
#define PTEXTSTYLES 0x74737874 /* "tsxt" */
#define PTRANSFERMODE 0x6D747070 /* "mtpp" */
#define PTRANSLATION 0x73727470 /* "srtp" */
#define PUNIFORMSTYLES 0x6C747375 /* "ltsu" */

```

```

#define PUPDATEON          0x64707570    /* "dpup" */
#define PUSERSELECTION     0x6C737570    /* "lsup" */
#define PVERSION           0x73726576    /* "srev" */
#define PVISIBLE           0x73697670    /* "sivp" */

#define pArcAngle          PARCANGLE
#define pBackgroundColor   PBACKGROUNDCOLOR
#define pBackgroundPattern PBACKGROUNDPattern
#define pBestType          PBESTTYPE
#define pBounds            PBOUNDS
#define pClass             PCLASS
#define pClipboard         PCLIPBOARD
#define pColor             PCOLOR
#define pColorTable        PCOLORTABLE
#define pContents          PCONTENTS
#define pCornerCurveHeight PCORNERCURVEHEIGHT
#define pCornerCurveWidth  PCORNERCURVEWIDTH
#define pDashStyle         PDASHSTYLE
#define pDefaultType       PDEFAULTTYPE
#define pDefinitionRect    PDEFINITIONRECT
#define pEnabled           PENABLED
#define pEndPoint          PENDPOINT
#define pFillColor         PFILLCOLOR
#define pFillPattern       PFILLPATTERN
#define pFont              PFONT
#define pFormula           PFORMULA
#define pGraphicObjects    PGRAPHICOBJECTS
#define pHasCloseBox       PHASCLOSEBOX
#define pHasTitleBar       PHASTITLEBAR
#define pID                PAEID
#define pIndex             PINDEX
#define pInsertionLoc      PINSERTIONLOC
#define pIsFloating        PISFLOATING
#define pIsFrontProcess    PISFRONTPROCESS
#define pIsModal           PISMODAL
#define pIsModified        PISMODIFIED
#define pIsResizable       PISRESIZABLE
#define pIsStationeryPad   PISSTATIONERYPAD
#define pIsZoomable        PISZOOMABLE
#define pIsZoomed          PISZOOMED
#define pItemNumber        PITEMNUMBER
#define pJustification      PJUSTIFICATION
#define pLineArrow         PLINEARROW
#define pMenuID            PMENUID
#define pName              PNAME
#define pNewElementLoc     PNEWELEMENTLOC
#define pPenColor          PPENCOLOR
#define pPenPattern        PPENPATTERN
#define pPenWidth          PPENWIDTH
#define pPixelDepth        PPIXELDEPTH
#define pPointList         PPOINTLIST
#define pPointSize         PPOINTSIZ
#define pProtection        PPROTECTION
#define pRotation          PROTATION
#define pScale             PSCALE

```


#define pScript	PSCRIPT
#define pSelected	PSELECTED
#define pSelection	PSELECTION
#define pStartAngle	PSTARTANGLE
#define pStartPoint	PSTARTPOINT
#define pTextColor	PTEXTCOLOR
#define pTextFont	PTEXTFONT
#define pTextItemDelimiters	PTEXTITEMDELIMITERS
#define pTextPointSize	PTEXTPOINTSIZE
#define pTextStyles	PTEXTSTYLES
#define pTransferMode	PTRANSFERMODE
#define pTranslation	PTRANSLATION
#define pUniformStyles	PUNIFORMSTYLES
#define pUpdateOn	PUPDATEON
#define pUserSelection	PUSERSELECTION
#define pVersion	PVERSION
#define pVisible	PVISIBLE

Type Descriptors

#define TYPEAETEXT	0x54585474	/* "TXTt" */
#define TYPEARC	0x63726163	/* "crac" */
#define TYPEBEST	0x74736562	/* "tseb" */
#define TYPECELL	0x6C656363	/* "lecc" */
#define TYPECLASSINFO	0x696C6367	/* "ilcg" */
#define TYPECOLORTABLE	0x74726C63	/* "trlc" */
#define TYPECOLUMN	0x6C6F6363	/* "locc" */
#define TYPEDASHSTYLE	0x73616474	/* "sadt" */
#define TYPEDATA	0x61746474	/* "atdt" */
#define TYPEDRAWINGAREA	0x77726463	/* "wrdc" */
#define TYPEELEMINFO	0x6E696C65	/* "nile" */
#define TYPEENUMERATION	0x6D756E65	/* "mune" */
#define TYPEEPS	0x20535045	/* "SPE" */
#define TYPEEVENTINFO	0x6E697665	/* "nive" */
#define TYPEFINDERWINDOW	0x6E697766	/* "niwf" */
#define TYPEFIXED	0x64786966	/* "dxif" */
#define TYPEFIXEDPOINT	0x746E7066	/* "tnpf" */
#define TYPEFIXEDRECTANGLE	0x74637266	/* "tcrf" */
#define TYPEGRAPHICCLINE	0x6E696C67	/* "nilg" */
#define TYPEGRAPHICTEXT	0x78746763	/* "xtgc" */
#define TYPEGROUPEDGRAPHIC	0x63697063	/* "cipc" */
#define TYPEINSERTIONLOC	0x6C736E69	/* "lsni" */
#define TYPEINTLTEXT	0x74787469	/* "txti" */
#define TYPEINTLWRITINGCODE	0x6C746E69	/* "ltni" */
#define TYPELONGDATETIME	0x2074646C	/* "tdl" */
#define TYPELONGFIXED	0x6478666C	/* "dxfl" */
#define TYPELONGFIXEDPOINT	0x7470666C	/* "tpfl" */
#define TYPELONGFIXEDRECTANGLE	0x6372666C	/* "crfl" */

```

#define TYPELONGPOINT          0x746E706C    /* "tnpl" */
#define TYPELONGRECTANGLE     0x7463726C    /* "tcrl" */
#define TYPEMACHINELOC        0x636F4C6D    /* "coLm" */
#define TYPEOVAL               0x6C766F63    /* "lvoc" */
#define TYPEPARAMINFO         0x6E696D70    /* "nimp" */
#define TYPEPICT               0x54434950    /* "TICP" */
#define TYPEPIXELMAP          0x78697063    /* "xipc" */
#define TYPEPIXMAPMINUS       0x6D6D7074    /* "mmpt" */
#define TYPEPOLYGON            0x6E677063    /* "ngpc" */
#define TYPEPROPINFO          0x666E6970    /* "fnip" */
#define TYPEQDPOINT            0x74704451    /* "tpDQ" */
#define TYPEQDRECTANGLE       0x74726471    /* "trdq" */
#define TYPERECTANGLE         0x63657263    /* "cerc" */
#define TYPERGB16              0x36317274    /* "6lrt" */
#define TYPERGB96              0x36397274    /* "69rt" */
#define TYPERGBCOLOR          0x42475263    /* "BGRc" */
#define TYPEROTATION           0x746F7274    /* "tort" */
#define TYPEROUNDEDRECTANGLE  0x63727263    /* "crrc" */
#define TYPEROW                0x776F7263    /* "worc" */
#define TYPESCRAPSTYLES       0x6C797473    /* "lyts" */
#define TYPESCRIPT             0x74706373    /* "tpcs" */
#define TYPESTYLEDTEXT         0x54585453    /* "TXTS" */
#define TYPESUITEINFO          0x6E697573    /* "nius" */
#define TYPETABLE              0x6C627463    /* "lbtc" */
#define TYPETEXTSTYLES         0x79747374    /* "ytst" */
#define TYPETIFF               0x46464954    /* "FFIT" */
#define TYPEVERSION            0x73726576    /* "srev" */

```

```

#define typeAEText             TYPEAETEXT
#define typeArc                 TYPEARC
#define typeBest                TYPEBEST
#define typeCell                TYPECELL
#define typeClassInfo           TYPECLASSINFO
#define typeColorTable          TYPECOLORTABLE
#define typeColumn              TYPECOLUMN
#define typeDashStyle           TYPEDASHSTYLE
#define typeData                TYPEDATA
#define typeDrawingArea         TYPEDRAWINGAREA
#define typeElemInfo            TYPEELEMINFO
#define typeEnumeration          TYPEENUMERATION
#define typeEPS                 TYPEEPS
#define typeEventInfo           TYPEEVENTINFO
#define typeFinderWindow        TYPEFINDERWINDOW
#define typeFixed               TYPEFIXED
#define typeFixedPoint          TYPEFIXEDPOINT
#define typeFixedRectangle      TYPEFIXEDRECTANGLE
#define typeGraphicLine         TYPEGRAPHICLINE
#define typeGraphicText         TYPEGRAPHICTEXT
#define typeGroupedGraphic      TYPEGROUPEDGRAPHIC
#define typeInsertionLoc        TYPEINSERTIONLOC
#define typeIntlText            TYPEINTLTEXT
#define typeIntlWritingCode     TYPEINTLWRITINGCODE
#define typeLongDateTime        TYPELONGDATETIME
#define typeLongFixed           TYPELONGFIXED
#define typeLongFixedPoint      TYPELONGFIXEDPOINT

```

#define typeLongFixedRectangle	TYPELONGFIXEDRECTANGLE
#define typeLongPoint	TYPELONGPOINT
#define typeLongRectangle	TYPELONGRECTANGLE
#define typeMachineLoc	TYPEMACHINELOC
#define typeOval	TYPEOVAL
#define typeParamInfo	TYPEPARAMINFO
#define typePict	TYPEPICT
#define typePixelMap	TYPEPIXELMAP
#define typePixMapMinus	TYPEPIXMAPMINUS
#define typePolygon	TYPEPOLYGON
#define typePropInfo	TYPEPROPINFO
#define typeQDPoint	TYPEQDPOINT
#define typeQDRectangle	TYPEQDRECTANGLE
#define typeRectangle	TYPERECTANGLE
#define typeRGB16	TYPERGB16
#define typeRGB96	TYPERGB96
#define typeRGBColor	TYPERGBCOLOR
#define typeRotation	TYPEROTATION
#define typeRoundedRectangle	TYPEROUNDEDRECTANGLE
#define typeRow	TYPEROW
#define typeScrapStyles	TYPESCRAPSTYLES
#define typeScript	TYPESCRIPT
#define typeStyledText	TYPESTYLEDTEXT
#define typeSuiteInfo	YPESUITEINFO
#define typeTable	TYPETABLE
#define typeTextStyles	TYPETEXTSTYLES
#define typeTIFF	TYPETIFF
#define typeVersion	TYPEVERSION

Errors

#define ERRABADKEYFORM	-10002
#define ERRACANTHANDLECLASS	-10010
#define ERRACANTSUPPLYTYPE	-10009
#define ERRACANTUNDO	-10015
#define ERRAEVENTFAILED	-10000
#define ERRAEINDEXTOOLARGE	-10007
#define ERRAEINTRANSACTION	-10011
#define ERRAELOCALONLY	-10016
#define ERRAEINOSUCHTRANSACTION	-10012
#define ERRAEINOTANELEMENT	-10008
#define ERRAEINOTASINGLEOBJECT	-10014
#define ERRAEINOTMODIFIABLE	-10003
#define ERRAEINOSUSERSELECTION	-10013
#define ERRAEINPRIVILEGEERROR	-10004
#define ERRAEINREADDENIED	-10005
#define ERRAEINTYPEERROR	-10001
#define ERRAEINWRITEDENIED	-10006

#define errAEBadKeyForm	ERRAEBADKEYFORM
#define errAECantHandleClass	ERRAECANTHANDLECLASS
#define errAECantSupplyType	ERRAECANTSUPPLYTYPE
#define errAECantUndo	ERRAECANTUNDO
#define errAEEEventFailed	ERRAEEVENTFAILED
#define errAEIndexTooLarge	ERRAEINDEXTOOLARGE
#define errAEInTransaction	ERRAEINTRANSACTION
#define errAELocalOnly	ERRAELOCALONLY
#define errAENoSuchTransaction	ERRAENOSUCHTRANSACTION
#define errAENotAnElement	ERRAENOTANELEMENT
#define errAENotASingleObject	ERRAENOTASINGLEOBJECT
#define errAENotModifiable	ERRAENOTMODIFIABLE
#define errAENoUserSelection	ERRAENOUSERSELECTION
#define errAEPrivilegeError	ERRAEPRIVILEGEERROR
#define errAEReadDenied	ERRAEREADDENIED
#define errAETypeError	ERRAETYPEERROR
#define errAEWriteDenied	ERRAEWRITEDENIED

Enumerations

```
enum {
    kBySmallIcon      = 0,
    kByIconView       = 1,
    kByNameView       = 2,
    kByDateView       = 3,
    kBySizeView       = 4,
    kByKindView       = 5,
    kByCommentView    = 6,
    kByLabelView      = 7,
    kByVersionView    = 8
};

enum {
    kAEInfo           = 11,
    kAEMain           = 0,
    kAESharing        = 13
};

enum {
    zoomIn            = 7,
    zoomOut           = 8
};
```

Types and Keys

```

#define KEYCOMPEVTTPARAMS      0x736D7270    /* "smrp" */
#define KEYEVENTCLASSPARAM     0x6C637665    /* "lcve" */
#define KEYEVENTIDPARAM        0x64697665    /* "dive" */

#define KSWITCHDESCTYPE        0x68637773    /* "hcws" */
#define KKEYEXMNCCLASS         0x6E6D7865    /* "nmxe" */
#define KEXMNKEY1              0x656E6F65    /* "enoe" */
#define KEXMNKEY2              0x6F777465    /* "owte" */

#define TYPECOMPEVTDESCRIPTOR   0x65706D63    /* "epmc" */
#define TYPECOMPEVTTPARAMS     0x736D7270    /* "smrp" */
#define TYPEWHOSERANGEINTERNAL  0x2A646E69    /* "*dni" */

```

Logical Operator Constants

```

#define KAEAND                  0x20444E41    /* " DNA" */
#define KAEOR                   0x2020524F    /* " RO" */
#define KAENOT                  0x20544F4E    /* " TON" */

```

Absolute ordinal constants

```

#define KAEFIRST                0x73726966    /* "srif" */
#define KAELAST                 0x7473616C    /* "tsal" */
#define KAEMIDDLE               0x6464696D    /* "ddim" */
#define KAEANY                  0x20796e61    /* " yna" */
#define KAEALL                  0x206C6c61    /* " lla" */

```

Relative Ordinal Constants

```

#define KAENEXT                 0x7478656E    /* "txen" */
#define KAEPREVIOUS             0x76657270    /* "verp" */

```

Keyword Constants

```
#define KEYAECOMPOPERATOR      0x6F6C6572  /* "oler" */
#define KEYAELOGICALTERMS     0x6D726574  /* "mret" */
#define KEYAELOGICALOPERATOR  0x63676F6C  /* "cgol" */
#define KEYAEOBJECT1          0x316A626F  /* "1jbo" */
#define KEYAEOBJECT2          0x326A626F  /* "2jbo" */
```

Keywords for Getting Fields out of Object Specifier Records

```
#define KEYAEDESIREDCLASS      0x746E6177  /* "tnaw" */
#define KEYAECONTAINER        0x6D6F7266  /* "morf" */
#define KEYAEKEYFORM          0x6D726F66  /* "mrof" */
#define KEYAEKEYDATA          0x646C6573  /* "dles" */
```

Keywords for Getting Fields out of Range Specifier Records

```
#define KEYAERANGESTART        0x72617473  /* "rats" */
#define KEYAERANGESTOP         0x706F7473  /* "pots" */
```

Special Handler Selectors for OSL Callbacks

```
#define KEYDISPOSETOKENPROC    0x6B6F7478  /* "kotx" */
#define KEYAECOMPAREPROC       0x72706D63  /* "rpmc" */
#define KEYAECOUNTPROC         0x746E6F63  /* "tnoc" */
#define KEYAEMARKTOKENPROC     0x64696B6D  /* "dikm" */
#define KEYAEMARKPROC          0x6B72616D  /* "kram" */
#define KEYAEADJUSTMARKSPROC   0x6D6A6461  /* "mjda" */
#define KEYAEGETERRDESCPROC    0x63646E69  /* "cdni" */
```

Value and Type Constants

keyAEKeyForm Possible Values

```
#define FORMUNIQUEID           0x20204449  /* " DI" */
#define FORMABSOLUTEPOSITION    0x78646E69  /* "xdni" */
#define FORMRELATIVEPOSITION    0x656C6572  /* "eler" */
#define FORMTEST                0x74736574  /* "tset" */
#define FORMRANGE               0x676E6172  /* "gnar" */
#define FORMPROPERTYID          0x706F7270  /* "porp" */
```

```
#define FORMNAME 0x656D616E /* "eman" */
```

Types

```
#define TYPEOBJECTSPECIFIER 0x206A626F /* "jbo" */
#define TYPEOBJECTBEINGEXAMINED 0x6E6D7865 /* "nmxe" */
#define TYPECURRENTCONTAINER 0x746E6363 /* "tncc" */
#define TYPETOKEN 0x656B6F74 /* "ekot" */
#define TYPERELATIVEDESCRIPTOR 0x206C6572 /* "ler" */
#define TYPEABSOLUTEORDINAL 0x6F736261 /* "osba" */
#define TYPEINDEXDESCRIPTOR 0x65646E69 /* "edni" */
#define TYPERANGEDESCRIPTOR 0x676E6172 /* "gnar" */
#define TYPELOGICALDESCRIPTOR 0x69676F6C /* "igol" */
#define TYPECOMPDESCRIPTOR 0x64706D63 /* "dpmc" */
```

Special Constants for Custom Whose-Clause Resolution

```
#define TYPEWHOSEDESCRIPTOR 0x736F6877 /* "sohw" */
#define FORMWHOSE 0x736F6877 /* "sohw" */
#define TYPEWHOSERANGE 0x676E7277 /* "gnrw" */
#define KEYAEWHOSERANGESTART 0x72747377 /* "rtsw" */
#define KEYAEWHOSERANGESTOP 0x70747377 /* "ptsw" */
#define KEYAEINDEX 0x7864696B /* "xdik" */
#define KEYAETEST 0x7473746B /* "tstk" */
```

Possible values for flags parameter to AEResolve. They are additive.

```
#define KAEIDOMINIMUM 0x0000
#define KAEIDOWHOSE 0x0001
#define KAEIDOMARKING 0x0004
```

OSL Error Codes

```
#define ERRAEIMPOSSIBLERANGE -1720
#define ERRAEWRONGNUMBERARGS -1721
#define ERRAEACCESSORNOTFOUND -1723

#define ERRAEENOSUCHLOGICAL -1725
#define ERRAEBADTESTKEY -1726
```

```
#define ERRAENOTANOBJSPEC      -1727
#define ERRAENOSUCHOBJECT      -1728

#define ERRAENEGATIVECOUNT    -1729
#define ERRAEMPTYLISTCONTAINER -1730
```

Types and Keys

```
#define keyCompEvtParams      KEYCOMPEVTPARAMS
#define keyEventClassParam   KEYEVENTCLASSPARAM
#define keyEventIDParam      KEYEVENTIDPARAM

#define kSwitchDescType      KSWITCHDESCTYPE
#define kKeyExmnClass        KKEYEXMNCLASS
#define kExmnKey1            KEXMNKEY1
#define kExmnKey2            KEXMNKEY2

#define typeCompEvtDescriptor TYPECOMPEVTDESCRIPTOR
#define typeCompEvtParams    TYPECOMPEVTPARAMS
#define typeWhoseRangeInternal TYPEWHOSERANGEINTERNAL
```

Logical Operator Constants

```
#define kAEAND      KAEAND
#define kAEOR       KAEOR
#define kAENOT      KAENOT
```

Absolute Ordinal Constants

```
#define kAEFirst      KAEFIRST
#define kAELast       KAELAST
#define kAEMiddle     KAEMIDDLE
#define kAEAny        KAEANY
#define kAEAll        KAEALL
```

Relative Ordinal Constants

```
#define kAENext          KAENEXT
#define kAEPrevious     KAEPREVIOUS
```

Keyword Constants

```
#define keyAECmpOperator    KEYAECOMPOPERATOR
#define keyAELogicalTerms  KEYAELOGICALTERMS
#define keyAELogicalOperator KEYAELOGICALOPERATOR
#define keyAEObject1       KEYAEOBJECT1
#define keyAEObject2       KEYAEOBJECT2
```

Keywords for getting fields out of object specifier records

```
#define keyAEDesiredClass  KEYAEDESIREDCLASS
#define keyAEContainer     KEYAECONTAINER
#define keyAEKeyForm       KEYAEKEYFORM
#define keyAEKeyData       KEYAEKEYDATA
```

Keywords for getting fields out of Range specifier records

```
#define keyAERangeStart    KEYAERANGESTART
#define keyAERangeStop     KEYAERANGESTOP
```

Special handler selectors for OSL Callbacks

```
#define keyDisposeTokenProc KEYDISPOSETOKENPROC
#define keyAECompareProc    KEYAECOMPAREPROC
#define keyAECntProc        KEYAECOUNTPROC
#define keyAEMarkTokenProc  KEYAEMARKTOKENPROC
#define keyAEMarkProc       KEYAEMARKPROC
#define keyAEAdjustMarksProc KEYAEADJUSTMARKSPROC
#define keyAEGetErrDescProc KEYAEGETERRDESCPROC
```

Value and Type Constants

keyAEKeyForm possible values

```
#define formUniqueID          FORMUNIQUEID
#define formAbsolutePosition  FORMABSOLUTEPOSITION
#define formRelativePosition  FORMRELATIVEPOSITION
#define formTest              FORMTEST
#define formRange              FORMRANGE
#define formPropertyID        FORMPROPERTYID
#define formName               FORMNAME
```

Types

```
#define typeObjectSpecifier    TYPEOBJECTSPECIFIER
#define typeObjectBeingExamined TYPEOBJECTBEINGEXAMINED
#define typeCurrentContainer   TYPECURRENTCONTAINER
#define typeToken              TYPETOKEN
#define typeRelativeDescriptor TYPERELATIVEDESCRIPTOR
#define typeAbsoluteOrdinal    TYPEABSOLUTEORDINAL
#define typeIndexDescriptor     TYPEINDEXDESCRIPTOR
#define typeRangeDescriptor     TYPERANGEDESCRIPTOR
#define typeLogicalDescriptor   TYPELOGICALDESCRIPTOR
#define typeCompDescriptor     TYPECOMPDESCRIPTOR
```

Special constants for custom whose-clause resolution

```
#define typeWhoseDescriptor    TYPEWHOSEDESCRIPTOR
#define formWhose              FORMWHOSE
#define typeWhoseRange         TYPEWHOSERANGE
#define keyAEWhoseRangeStart   KEYAEWHOSERANGESTART
#define keyAEWhoseRangeStop    KEYAEWHOSERANGESTOP
#define keyAEIndex             KEYAEINDEX
#define keyAETest              KEYAETEST
```

Possible values for flags parameter to AEResolve. They are additive.

```
#define kAEIDoMinimum          KAEIDOMINIMUM
#define kAEIDoWhose             KAEIDOWHOSE
#define kAEIDoMarking           KAEIDOMARKING
```

OSL Error Codes

```
#define errAEImpossibleRange  ERRAEIMPOSSIBLERANGE
#define errAEWrongNumberArgs  ERRAEWRONGNUMBERARGS
```

```
#define errAEAccessorNotFound      ERRAEACCESSORNOTFOUND

#define errAENoSuchLogical        ERRAENOSUCHLOGICAL
#define errAEBadTestKey           ERRAEBADTESTKEY

#define errAENotAnObjSpec         ERRAENOTANOBJSPEC
#define errAENoSuchObject         ERRAENOSUCHOBJECT

#define errAENegativeCount        ERRAENEGATIVECOUNT
#define errAEEmptyListContainer   ERRAEEMPTYLISTCONTAINER
```

Return Codes by Number

This section lists OSA errors in order of their error number and provides an explanation for each error.

0
1
2
3
4
8
87
122
127
99
-1700
-1700
-1701
-1702
-1703
-1704
-1705
-1706
-1707
-1708
-1709
-1710
-1711
-1712
-1713
-1714
-1715
-1716
-1717
-1718
-1719
-1720
-1721
-1723
-1725
-1726
-1727
-1728
-1729
-1730
-1731
-1732
-1733
-1734
-1735
-1750
-1751
-1752
-1753

-1754
-1756
-1757
-1758
-1759
-1761
-1762
-3005
-3006
-3007
-3008
-3009
-10000
-10001
-10002
-10003
-10004
-10005
-10006
-10007
-10008
-10009
-10010
-10011
-10012
-10013
-10014
-10015
-10016

Return Codes by Name

This section lists OSA errors in order of their error name and provides an explanation for each error.

errAEAccessorNotFound
errAEAppAlreadyInstalled
errAEAppNotFound
errAEBadKeyForm
errAEBadListItem
errAEBadParm
errAEBadTestKey
errAEBufferTooSmall
errAECantHandleClass
errAECantSupplyType
errAECantUndo
errAEC coercionFail
errAECorruptData
errAEDescNotFound
errAEEEmptyListContainer
errAEEEventFailed
errAEEEventNotHandled
errAEHandlerNotFound
errAEIllegalIndex
errAEImpossibleRange
errAEIndexTooLarge
errAEInTransaction
errAELocalOnly
errAEMgrNotInitialized
errAENegativeCount
errAENewerVersion
errAENoSuchLogical
errAENoSuchObject
errAENoSuchTransaction
errAENotAEDesc
errAENotAnElement
errAENotAnObjSpec
errAENotASignalObject
errAENotASpecialFunction
errAENotModifiable
errAENotOSAEvent

errAENoUserInteraction
errAEParmMissed
errAENoUserSelection
errAEPrivilegeError
errAEReadDenied
errAERecordingIsAlreadyOn
errAEReplyNotArrived
errAEReplyNotValid
errAEResourceNotFound
errAESysHandlerNotLoaded
errAESystemError
errOSASystemError
errAETimeout
errAETypeError
errAEUnknownAddrfieldsType
errAEUnknownObjectType
errAEUnknownSendMode
errAEMWaitCanceled
errAEWriteDenied
errAEWrongDataType
errAEWrongNumberArgs
errCMInvalidComponentID
errCMInvalidComponentInstance
errCMComponentAlreadyInstalled
errCMBadParm
errCMSystemError
errOSABadSelector
errOSABadStorageType
errOSACantCoerce
errOSACantOpenComponent
errOSAComponentMismatch
errOSADDataFormatObsolete
errOSADDataFormatTooNew
errOSAInvalidID
errOSANoSuchDialect
errOSAScriptError
errOSASourceNotAvailable
ERROR_INSUFFICIENT_BUFFER
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY
ERROR_PROC_NOT_FOUND
noErr

0

0

noErr
No error.

1

1

errAEAppAlreadyInstalled
The application or part handler is already in the registration data base.

2

2

errAEAppNotFound

The specified application or part handler is not found in the registration data base.

3

3

errAEBufferTooSmall

The size of the buffer is not large enough to hold the data to be returned.

4

4

errAEResourceNotFound

The requested resource is not found in the registration data base.

8

8

ERROR_NOT_ENOUGH_MEMORY

There is not enough memory.

87

87

ERROR_INVALID_PARAMETER

One or more of the parameters passed in are invalid.

122

122

ERROR_INSUFFICIENT_MEMORY

There is not enough memory.

127

127

ERROR_PROC_NOT_FOUND

There is no eligible process with the specified PID.

99

99

errAEBadParm

One or more of the parameters passed in are invalid.

-1700

-1700

errAEC coercionFail

Data could not be coerced to the requested descriptor type.

-1700a

-1700a

errAECantCoerce

Data could not be coerced to the requested descriptor type.

-1701

-1701

errAEDescNotFound

Descriptor record was not found.

-1702

-1702

errAECorruptData

Data in an OSA event could not be read.

-1703

-1703 **errAEWrongDataType**
Wrong descriptor type.

-1704

-1704 **errAENotAEDesc**
Not a valid descriptor record.

-1705

-1705 **errAEBadListItem**
Operation involving a list item failed.

-1706

-1706 **errAENewerVersion**
Need a newer version of the OSA Event Manager.

-1707

-1707 **errAENotOSAEvent**
Event is not an OSA event.

-1708

-1708 **errAEEventNotHandled**
Event was not handled by an OSA event handler.

-1709

-1709

errAEReplyNotValid

[AEResetTimer](#) was passed an invalid reply.

-1710

-1710

errAEUnknownSendMode

Invalid sending mode was passed.

-1711

-1711

errAEWaitCanceled

User canceled out of wait loop for reply or receipt.

-1712

-1712

errAETimeout

OSA event timed out.

-1713

-1713

errAENoUserInteraction

No user interaction is allowed.

-1714

-1714

errAENotASpecialFunction

The keyword is not valid for a special function.

-1715

-1715

errAEParmMissed

Handler cannot understand a parameter the client considers required.

-1716

-1716

errAEUnknownAddressType

Unknown OSA event address type.

-1717

-1717

errAEHandlerNotFound

No handler found for an OSA event or a coercion, or no object callback function found.

-1718

-1718

errAEReplyNotArrived

Reply has not yet arrived.

-1719

-1719

errAEIllegalIndex

Not a valid list index.

-1720

-1720

errAEImpossibleRange

The range is not valid because it is impossible for a range to include the first and last objects that were specified; an example is a range in which the offset of the first object is greater than the offset of the last object.

-1721

-1721

errAEWrongNumberArgs

The number of operands provided for the kAENOT logical operator is not 1.

-1723

-1723

errAEAccessorNotFound

There is no object accessor function for the specified object class and token descriptor type.

-1725

-1725

errAENoSuchLogical

The logical operator in a logical descriptor record is not kAEAND, kAEOR, or kAENOT.

-1726

-1726

errAEBadTestKey

The descriptor record in a test key is neither a comparison descriptor record nor a logical descriptor record.

-1727

-1727

errAENotAnObjSpec

The *objectSpecifier* parameter of [AEResolve](#) is not an object specifier record.

-1728

-1728

errAENoSuchObject

A run-time resolution error; for example, object specifier record asked for the third element, but there are only 2.

-1729

-1729

errAENegativeCount

Object-counting function returned negative value.

-1730

-1730

errAEEmptyListContainer

The container for an OSA event object is specified by an empty list.

-1731

-1731

errAEUnknownObjectType

Descriptor type of token returned by [AEResolve](#) is not known to server application

-1732

-1732

errAERecordingIsAlreadyOn

Attempt to turn recording on when it is already on

-1733

-1733

errAEMgrNotInitialized

AEInit has not been executed.

-1734

-1734

errAESysHandlerNotLoaded

The module for a sysstem handler could not be loaded.

-1735

-1735

errAESystemError

An OSA Event Manager system error occurred.

-1750

-1750

errOSASystemError

A general scripting system error occurred.

-1751

-1751

errOSAInvalidID

An invalid script ID is specified.

-1752

-1752

errOSABadStorageType

Illegal storage type.

-1753

-1753

errOSAScriptError

Error occurred during compilation or execution.

-1754

-1754 **errOSABadSelector**
Selector not supported by scripting component.

-1756

-1756 **errOSASourceNotAvailable**
Source data not available

-1757

-1757 **errOSANoSuchDialect**
Invalid dialect code.

-1758

-1758 **errOSADataFormatObsolete**
Data format is obsolete.

-1759

-1759 **errOSADataFormatTooNew**
Data format is too new.

-1761

-1761 **errOSAComponentMismatch**

Generic scripting component error; parameters are for two different scripting components instead of the same one.

-1762

-1762

errOSACantOpenComponent

Generic scripting component error; cannot connect to scripting component.

-2700

-2700

errOSAGeneralError

An error has occurred.

-2701

-2701

errOSADivideByZero

An attempt was made to divide by zero.

-2702

-2702

errOSANumericOverflow

An integer or real value is too large to be represented.

-2703

-2703

errOSACantLaunch

An application cannot be launched.

-2704

-2704

errOSAAppNotHighLevelEventAware

An application cannot respond to an OSA event.

-2705

-2705

errOSACorruptTerminology

An application's terminology resource is not readable.

-2707

-2707

errOSAInternalTableOverflow

A run-time internal data structure overflowed.

-2708

-2708

errOSADataBlockTooLarge

-2709

-2709

errOSACantGetTerminology

-2710

-2710

errOSACantCreate

-2740

-2740

errOSASyntaxError

A syntax error occurred.

-2741

-2741

errOSASyntaxTypeError

Another form of syntax was expected; for example, if <type> was expected but <this> was found.

-2742

-2742

errOSATokenTooLong

A name or number is too long to be parsed.

-2750

-2750

errOSADuplicateParameter

A formal parameter, local variable, or instance variable is specified more than once.

-2751

-2751

errOSADuplicateProperty

A formal parameter, local variable, or instance variable is specified more than once.

-2752

-2752

errOSADuplicateHandler

More than one handler is defined with the same name in a scope where the language does not allow.

-2753

-2753

errOSAUndefinedVariable

A variable is accessed that has no value.

-2754

-2754

errOSAInconsistentDeclarations

A variable is declared inconsistently in the same scope; for example, both local and global.

-3005

-3005

errCMInvalidComponentID

The component ID is not found in the registration data base.

-3006

-3006

errCMInvalidComponentInstance

The component instance is invalid.

-3007

-3007

errCMComonentAlreadyInstalled

The specified component is already in the registration data base.

-3008

-3008

errCMBadParm

One or more of the parameters passed in are invalid.

-3009

-3009

errCMSYSTEMError

The Component Manager encountered a system error.

-2755

-2755

errOSACONTROLFlowError

A variable is declared inconsistently in the same scope; for example, both local and global.

10000

-10000

errAEEVENTFailed

10001

-10001

errAETYPEError

10002

-10002

errAEBADKeyForm

10003

-10003

errAENOTModifiable

10003

-10003

errOSAILLEGALAssign

An object can never be set in a container.

10004

-10004 **errAEPrivilegeError**

10005

-10005 **errAEReadDenied**

10006

-10006 **errAEWriteDenied**

An object cannot be set in a container.

10006

-10006 **errOSACantAssign**

An object cannot be set in a container.

10007

-10007 **errAEIndexTooLarge**

10008

-10008 **errAENotAnElement**

10009

-10009

errAECantSupplyType

10010

-10010

errAECantHandleClass

10011

-10011

errAEInTransaction

10012

-10012

errAENoSuchTransaction

10013

-10013

errAENoUserSelection

10014

-10014

errAENotASingleObject

10015

-10015

errAECantUndo

10016

-10016

errAELocalOnly

Presentation Manager Messages Related to OSA

This section provides information about messages that can be processed by windows and dialog procedures within OS/2 applications.

WM_SEMANTICEVENT

WM_SEMANTICEVENT Field - pSemanticEvent

pSemanticEvent ([pSemanticEvent](#))
Pointer to a semantic event record.

WM_SEMANTICEVENT Field - ulReserved

ulReserved (long)
Reserved value; should be 0.

WM_SEMANTICEVENT - Parameters

pSemanticEvent ([pSemanticEvent](#))
Pointer to a semantic event record.

ulReserved (long)
Reserved value; should be 0.

WM_SEMANTICEVENT - Syntax

This message occurs when a semantic event is received.

```
param1
    pSemanticEvent  pSemanticEvent

param2
    long            ulReserved
```

WM_SEMANTICEVENT - Remarks

Applications that do not support semantic events can pass all events to the [AEProcessOSAEvent](#) method.

WM_SEMANTICEVENT - Default Processing

The default window procedure takes no action on this message.

WM_SEMANTICEVENT - Examples

The following pseudocode shows how the OSA Event Manager is used in a typical PM application, but does not show exception handling.

```
hab = WinInitialize();
hmq = WinCreateMsgQueue(hab, 0);
WinRegisterClass(hab, ..., WndProc, ...);
hwndFrame = WinCreateStandardWindow(..., &hwndClient);

AEInit (hwndClient, argv[0], TRUE);
AEInstallEventHandler(...);

while (WinGetMsg (hab, &qmsg, ...))
    WinDispatchMsg(hab, &qmsg);

MRESULT EXPENTRY WndProc (HWND hwnd, ULONG msg, ....) {

    switch (msg) {

        case WM_SEMANTICEVENT:
            err = AEProcessOSA Event (...);
            /* error handling */
            break;

    }
    return WinDefWindowProc (hwnd, msg, mp1, mp2);
}
AETerminate();
```

WM_SEMANTICEVENT - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Remarks](#)

[Default Processing](#)

[Examples](#)

[Glossary](#)

Object REXX OSA Scripting Component Information

This appendix provides supplemental Object REXX OSA Scripting Component information not available elsewhere in this book.

Scripting Component Registration

The Object REXX Scripting Component will be registered with the following values:

Component Type	osa
Component Subtype	OREX
Component Manufacturer	IBM
Component Flags	kOSASupportsCompiling kOSASupportsConvenience kOSASupportsGetSource
Component Name	Object REXX
Component Info	Object REXX Scripting Component

The following groups of routines will not be supported by Object REXX in this release of OS/2.

- kOSASupportsAECOercion
 - kOSASupportsAESending
 - kOSASupportsDialects
 - kOSASupportsEventHandling
 - kOSASupportsRecording
-

Supported Scripting Component Methods

The following table lists the Scripting Component methods supported by Object REXX.

Flag	Method
Required	OSADisplay OSADispose OSAExecute OSAGetActiveProc OSAGetScriptInfo OSALoad OSAScriptError OSASetActiveProc OSASetScriptInfo OSAStore

Compiling	OSACompile OSACopyID OSAScriptingComponentName
Convenience	OSALoadExecute OSACompileExecute OSADoScript
GetSource	OSAGetSource

The following Object REXX methods are available to the ScriptableApp class:

Methods the ScriptableApp Class Defines

[CONNECT](#)
[DISCONNECT](#)

The following methods are inherited from the Object class:

NEW (Class method)
 =
 ==
 \=
 ><
 <>
 \==
 CLASS
 COPY
 DEFAULTNAME
 HASMETHOD
 INIT
 OBJECTNAME
 OBJECTNAME=
 REQUEST
 RUN
 SETMETHOD
 START
 STRING
 UNSETMETHOD

Note: If the method dictionary of the scriptable application contains a method with the same name as one of these methods, this method overrides the Object class method.

The *Object REXX Reference for OS/2* describes the Object class methods in detail.

Object REXX Methods Available to the Object Specifier Class

The following methods are defined in the Object Specifier class:

[]
 >> (Corresponds to OSA operator "begins with")
 << (Corresponds to OSA operator "ends with")
 == (Corresponds with OSA operator "contains")

The following methods are inherited from the Object class:

CLASS
 COPY
 HASMETHOD
 INIT
 OBJECTNAME
 OBJECTNAME=
 REQUEST
 RUN
 SETMETHOD
 START
 UNSETMETHOD

The *Object REXX Reference for OS/2* describes the Object class methods in detail.

Notices

August 1996

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

(C) Copyright International Business Machines Corporation 1996. All rights reserved.

(C) Copyright Apple Computers, Inc 1994. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
Common User Access
CUA
IBM
OS/2
Personal System/2
Presentation Manager
SAA
System Applications Architecture
Workplace Shell

The following terms are trademarks of other companies:

1-2-3	Lotus Development Corporation
Apple	Apple Computer, Inc.
C++	American Telephone and Telegraph Company
Display PostScript	Adobe Systems Incorporated
Helvetica	Linotype Company
Lotus	Lotus Development Corporation
Macintosh	Apple Computer, Inc.
OpenDoc	Apple Computer, Inc.
Palantino	Linotype Company
Windows	Microsoft Corporation

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

This glossary defines many of the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing*, as well as terms specific to the OS/2 operating system and the Presentation Manager. It is not a complete glossary for the entire OS/2 operating system; nor is it a complete dictionary of computer terms.

Other primary sources for these definitions are:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

Select a starting letter of glossary terms:

0-9

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

Glossary - 0-9

8.3 file-name format -A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format* .

Glossary - A

abstract class -A class used only to derive other classes. An abstract class is never instantiated. Contrast with *concrete class* .

abstract superclass -A superclass listed in the *OSA Event Registry: Standard Suites* , such as cObject or cOpenableObject, that is used only in definitions of object classes and not for real OSA event objects. See also *object class* .

accelerator -In SAA Common User Access architecture, a key or combination of keys that invokes an application-defined function.

accelerator table -A table used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access mode -The manner in which an application gains access to a file it has opened. Examples of access modes are read-only, write-only, and read/write.

access permission -All access rights that a user has regarding an object. (I)

action -One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from a menu bar or menu.

action data -Information stored in the undo object's action history that allows a part to reverse the effects of an undoable action.

action history -The cumulative set of reversible actions available at any one time, maintained by the undo object.

action subhistory -A subset of action data added to the undo object's action history by a part in a modal state. The part can then remove the subhistory from the action history without affecting earlier actions.

action type -A constant that defines whether an undoable action is a single-stage action (such as a cut) or part of a two-stage action (such as a drag-move).

activate -(1) For a part, to make ready to receive the selection focus. A frame is activated when a mouse-down event occurs within it. (2) For a window, to bring it to the front by passing the cursor over it.

active frame -The frame that has the selection focus and usually the keyboard focus. Editing takes place in the active frame; the selection or insertion point is displayed within the frame. The active frame usually has the keystroke and menu foci, also.

active part -The part displayed in the active frame. The active part controls the part-specific palettes and menus, and its content contains the selection or insertion point. The active part can be displayed in one or more frames, only one of which is the active frame.

active program -A program currently running on the computer. An active program can be interactive (running and receiving input from the user) or noninteractive (running but not receiving input from the user). See also *interactive program* and *noninteractive program* .

active shape -A shape that describes the portion of a facet within which a part expects to receive user events. If, for example, an embedded part's used shape and active shape are identical, the containing part both draws and accepts events in the unused areas within the embedded part's frame.

active window -The window with which the user is currently interacting.

additional parameter -A keyword-specified descriptor record that a server application uses in addition to the data specified in the direct parameter. For example, an OSA event for arithmetic operations may include additional parameters that specify operands in an equation. Additional parameters may be required, or they may be optional.

address descriptor record -A descriptor record of data type AAddressDesc that contains the address of the target or source of an OSA event.

address space -(1) The range of addresses available to a program. (A) (2) The area of virtual storage available for a particular job.

alphanumeric video output -Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is

addressed by an application as a rectangular array of character cells.

AE record -A descriptor record of data type AERecord that usually contains a list of parameters for an OSA event. See also *OSA event parameter* .

American National Standard Code for Information Interchange -The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

ancestor -See *superclass* .

anchor -A window procedure that handles Presentation Manager* message conversions between an icon procedure and an application.

anchor block -An area of Presentation-Manager-internal resources to allocated process or thread that calls WinInitialize.

anchor point -A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

annotation -A property in a part's storage unit that is separate from the part's contents.

ANSI -American National Standards Institute.

APA -All points addressable.

API -Application programming interface.

application -A collection of software components used to perform specific types of user-oriented work on a computer; for example, a payroll application, an airline reservation application, a network application. See also *conventional application* .

application-modal -Pertaining to a message box or dialog box for which processing must be completed before further interaction with any other window owned by the same application may take place.

application object -In SAA Advanced Common User Access architecture, a form that an application provides for a user; for example, a spreadsheet form. Contrast with *user object* .

application programming interface (API) -A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application result handler -A result handler that is associated with a particular application. Contrast with *system result handler* .

arbitrator -An OpenDoc object that manages negotiation among parts about ownership of shared resources. Examples of such resources are the menu focus, selection focus, keystroke focus, and the serial ports.

area -In computer graphics, a filled shape such as a solid rectangle.

ASCII -American National Standard Code for Information Interchange.

ASCIIZ -A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio -In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNC) -(1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See also *synchronous* .

atom -A constant that represents a string. As soon as a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table* . See *integer atom* .

atom table -A table used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

atomic operation -An operation that completes its work on an object before another operation can be performed on the same object.

attribute -A characteristic or property that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes* .

automatic link -In Information Presentation Facility (IPF), a link that begins a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

auxiliary storage unit -An extra storage unit that a part uses to store its contents. Contrast with *main storage unit* .

AVIO -Advanced Video Input/Output.

Glossary - B

background -(1) In multiprogramming, the conditions under which low-priority programs are executed. Contrast with *foreground* . (2) An active session that is not currently displayed on the screen.

background color -The color in which the background of a graphic primitive is drawn.

background mix -An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix* .

background program -In multiprogramming, a program that executes with a low priority. Contrast with *foreground program* .

base class -See *superclass* .

base draft -The original draft of a document. Every OpenDoc document has a base draft, from which all subsequent drafts are ultimately derived. See also *current draft* .

base menu bar -The menu bar that contains the menus shared by all parts in a document. The document shell installs the base menu bar; parts add their own menus and items.

base object -The object whose interface is extended by an extension object.

Bento -A container suite that implements OpenDoc storage on OS/2 and some other platforms.

Bézier curve -(1) A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier. (2) In the AIX Graphics Library, a cubic spline approximation to a set of four control points that passes through the first and fourth control points and that has a continuous slope where two spline segments meet. Named after Dr. P. Bézier.

bias transform -A transform that is applied to measurements in a part's coordinate system to change them into *platform-normal coordinates* .

binding -(1) In programming, an association between a variable and a value for that variable that holds within a defined scope. The scope may be that of a rule, a function call or a procedure invocation. (2) In OpenDoc, the process of selecting an executable code module based on type information. (3) In SOM, a file enabling a compiler to match a method implementation with its declaration. Also called a header file.

bit map -A representation in memory of the data displayed on an APA device, usually the screen.

block -(1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (T) (2) To record data in a block. (3) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (A)

block device -A storage device that performs I/O operations on blocks of data called *sectors* . Data on block devices can be randomly accessed. Block devices are designated by a drive letter (for example, **C:**).

blocking mode -A condition set by an application that determines when its threads might block. For example, an application might set the Pipemode parameter for the DosCreateNPipe function so that its threads perform I/O operations to the named pipe block when no data is available.

border -(1) A visual indication (for example, a separator line or a background color) of the boundaries of a window. (2) For OpenDoc, see *frame border* .

boundary determination -An operation used to compute the size of the smallest rectangle that encloses a graphics object on the screen.

boundary objects -The elements, specified in a range descriptor record, that identify the beginning and end of the range. See also *range descriptor record* .

breakpoint -(1) A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for resuming execution. (T) (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broken pipe -When all of the handles that access one end of a pipe have been closed.

bucket -One or more fields in which the result of an operation is kept.

buffer -(1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers. (A)

bundled frame -A frame whose contents do not respond to user events. For example, a mouse click within a bundled frame selects but does not activate the frame.

button -A mechanism used to request or initiate an action. See also *barrel buttons* , *bezel buttons* , *mouse button* , *push button* , and *radio button* .

byte pipe -Pipes that handle data as byte streams. All unnamed pipes are byte pipes. Named pipes can be byte pipes or message pipes. See also *byte stream* .

byte stream -Data that consists of an unbroken stream of bytes.

Glossary - C

cache -A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space -A presentation space from a Presentation-Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

CAD -Computer-Aided Design.

call -(1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

calling sequence -A sequence of instructions together with any associated data necessary to execute a call. (T)

Cancel -An action that removes the current window or menu without processing it, and returns the previous window.

canvas -The platform-specific drawing environment on which frames are laid out. Each window or printing device has one drawing canvas. See also *static canvas* , *dynamic canvas* , and *drawing canvas* .

canvas coordinate space -The coordinate space of the canvas upon which a part's content is drawn. It may or may not be equal to *window coordinate space* .

cascaded menu -In the OS/2 operating system, a menu that appears when the arrow to the right of a cascading choice is selected. It contains a set of choices that are related to the cascading choice. Cascaded menus are used to reduce the length of a menu. See also *cascading choice* .

cascading choice -In SAA Common User Access architecture, a choice in a menu that, when selected, produces a cascaded menu containing other choices. An arrow () appears to the right of the cascading choice.

CASE statement -In PM programming, provides the body of a window procedure. There is usually one CASE statement for each message type supported by an application.

category -See *part category* .

CGA -Color graphics adapter.

chained list -A list in which the data elements may be dispersed but in which each data element contains information for locating the next. (T) Synonymous with *linked list* .

change ID -(1) A number used to identify a particular instance of clipboard contents. (2) A number used to identify a particular instance of link source data.

character -A letter, digit, or other symbol.

character box -In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode* . Contrast with *character cell* .

character cell -The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box* .

character code -The means of addressing a character in a character set, sometimes called *code point* .

character device -A device that performs I/O operations on one character at a time. Because character devices view data as a stream of bytes, character-device data cannot be randomly accessed. Character devices include the keyboard, mouse, and printer, and are referred to by name.

character mode -A mode that, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

character set -(1) An ordered set of unique representations called characters; for example, the 26 letters of English alphabet, Boolean 0 and 1, the set of symbols in the Morse code, and the 128 ASCII characters. (A) (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.

check box -In SAA Advanced Common User Access architecture, a square box with associated text that represents a choice. When a user selects a choice, an X appears in the check box to indicate that the choice is in effect. The user can clear the check box by selecting the choice again. Contrast with *radio button*.

check mark - (1) In SAA Advanced Common User Access architecture, a symbol that shows that a choice is currently in effect. (2) The symbol that is used to indicate a selected item on a pull-down menu.

child class -See *subclass*.

child process -In the OS/2 operating system, a process started by another process, which is called the parent process. Contrast with *parent process*.

child window -A window that appears within the border of its parent window (either a primary window or another child window). When the parent window is resized, moved, or destroyed, the child window also is resized, moved, or destroyed; however, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window*.

choice -(1) An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol). (2) In SAA Common User Access architecture, an item that a user can select.

chord -(1) To press more than one button on a pointing device while the pointer is within the limits that the user has specified for the operating environment. (2) In graphics, a short line segment whose end points lie on a circle. Chords are a means for producing a circular image from straight lines. The higher the number of chords per circle, the smoother the circular image.

CI Labs -See *Component Integration Laboratories*.

circular link -A configuration of links in which changes to a link's destination directly or indirectly affect its source.

class -In object-oriented design or programming, a group of objects that share a common definition and that therefore share common properties, operations, and behavior. Members of the group are called instances of the class.

class hierarchy -The structure by which classes are related through inheritance.

class method -In System Object Model, an action that can be performed on a class object. Synonymous with *factory method*.

class object -In System Object Model, the run-time implementation of a class.

class style -The set of properties that apply to every window in a window class.

client -(1) A functional unit that receives shared services from a server. (T) (2) A user, as in a client process that uses a named pipe or queue that is created and owned by a server process.

client application -An application that uses OSA events to request a service (for example, printing a list of files, checking the spelling of a list of words, or performing a numeric calculation) from another application (called a server application). These applications can reside on the same local computer or on remote computers connected to a network.

client area -The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client program -An application that creates and manipulates instances of classes.

client window -The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clip limits -The area of the paper that can be reached by a printer or plotter.

clip shape -(1) The structure by which classes are related through inheritance. (2) A shape that defines the limits of drawing within a facet.

clipboard -In SAA Common User Access architecture, an area of computer memory, or storage, that temporarily holds data. Data in the clipboard is available to other applications.

clipboard focus -In OpenDoc, a designation of ownership of access to the clipboard. The part with the clipboard focus can read from and write to the clipboard.

clipping -In computer graphics, removing those parts of a display image that lie outside a given boundary. (I) (A)

clipping area -The area in which the window can paint.

clipping path -A clipping boundary in world-coordinate space.

clock tick -The minimum unit of time that the system tracks. If the system timer currently counts at a rate of X Hz, the system tracks the time

every 1/X of a second. Also known as *time tick* .

CLOCK\$ -Character-device name reserved for the system clock.

clone -To copy an object and all its referenced objects. When you clone an object, that object plus all other objects to which there is a *strong persistent reference* in the cloned object are copied.

close -For a frame, to remove from memory but not from storage. A closed frame is not permanently removed from its document. Contrast with *remove* .

Code Fragment Manager (CFM) -The portion of system software that manages the run-time loading and dynamic linking of code modules.

code page -An assignment of graphic characters and control-function meanings to all code points.

code point -(1) Synonym for *character code* . (2) A 1-byte code representing one of 256 potential characters.

code segment -An executable section of programming code within a load module.

coercion handler -In the Open Scripting Architecture, a function that converts data from one descriptor type into another.

color dithering -See *dithering* .

color graphics adapter (CGA) -An adapter that simultaneously provides four colors and is supported by all IBM Personal Computer and Personal System/2 models.

command -The name and parameters associated with an action that a program can perform.

command area -An area composed of a command field prompt and a command entry field.

command entry field -An entry field in which users type commands.

command ID -A position-independent identifier for a menu command. See also *synthetic command ID* .

command line -On a display screen, a display line, sometimes at the bottom of the screen, in which only commands can be entered.

command mode -A state of a system or device in which the user can enter commands.

command prompt -A field prompt showing the location of the command entry field in a panel.

Common Object Request Broker Architecture (CORBA) -A standard promulgated by the Object Management Group industry consortium for defining interactions among objects.

Common Programming Interface (CPI) -Definitions of those application development languages and services that have, or are intended to have, implementations on and a high degree of commonality across the SAA environments. One of the three SAA architectural areas. See also *Common User Access architecture* .

Common User Access (CUA) architecture - Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas. See also *Common Programming Interface* .

comparison descriptor record -A coerced AE record of type typeCompDescriptor that specifies an OSA event object and either another OSA event object or data for the OSA Event Manager to compare to the first object.

compile -To translate a program written in a higher-level

compiled script file -A script file with the file type 'scpt' that contains script data as a resource of type 'scpt'. Before executing the script in a compiled script file, a user must first open the script from an application such as Script Editor.

component -(1) Hardware or software that is part of a functional unit. A functional part of an operating system; for example, the scheduler or supervisor. (2) A set of modules that performs a major function within a system; for example, a compiler or a master scheduler. (3) A software product that functions in the OpenDoc environment. Part editors, part viewers, and services are examples of components. See also *application component*, *service component* .

Component Integration Laboratories (CI Labs) -A consortium of platform and application vendors that oversees the development and distribution of OpenDoc technology.

component-specific storage descriptor record -A descriptor record returned by OSASStore. The descriptor type for a component-specific storage descriptor record is the scripting component subtype value for the scripting component that created the script data.

composite window -A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

compound document -A single document containing multiple, heterogeneous data types, each created, presented and edited by its own software. A compound document is made up of *parts* .

computer-aided design (CAD) -The use of a computer to design or change a product, tool, or machine, such as using a computer for

drafting or illustrating.

COM1, COM2, COM3 -Character-device names reserved for serial ports 1 through 3.

CON -Character-device name reserved for the console keyboard and screen.

concrete class -A class designed to be instantiated. Contrast with *abstract class* .

conditional cascaded menu -A pull-down menu associated with a menu item that has a cascade mini-push button beside it in an object's pop-up menu. The conditional cascaded menu is displayed when the user selects the mini-push button.

connect or reconnect -For a frame object, to reestablish its connection to the part it displays. Reconnecting a frame may involve recreating it from storage.

container -(1) In SAA Common User Access architecture, an object that holds other objects. A folder is an example of a container object. (2) A holder of persistent data (documents); part of the OpenDoc *container suite* . (3) An OSA event object that contains another OSA event object. A container is specified in an object specifier record by a keyword-specified descriptor record with the keyword *keyAECContainer*. The keyword-specified descriptor record is usually another object specifier record. It can also be a null descriptor record, or it can be used much like a variable when the OSA Event Manager determines a range or performs a series of tests. The objects a container contains can be either elements or properties. (See also *OSA event object* , *element* , *object specifier record* , *property* , *container part* , *folder* and *object* .

container hierarchy -The chain of containers that determine the location of one or more OSA event objects. See also *container* .

container part -A part that can embed other parts within its content. A container part is capable of being a *containing part* . Contrast with *simple part* and *noncontainer part* . See also *container application* .

containing frame -The display frame of an embedded frame's containing part. Each embedded frame has one containing frame; each containing frame has one or more embedded frames.

containing part -The part that immediately contains an embedded part. Each embedded part has one containing part; each containing part has one or more embedded parts.

container suite -A document storage architecture, built on top of a platform's native file system, that allows for the creation, storage, and retrieval of compound documents. A container suite is implemented as a set of OpenDoc classes: containers, documents, drafts, and storage units. See also *Bento* .

containment -A relationship between objects wherein an object of one class contains a reference to an object of another class. Contrast with *inheritance* .

content -See *part content* .

content area -The potentially visible area of a part as viewed in a frame or window. If the content area is greater than the area of the frame or window, only a portion of the part can be viewed at a time.

content coordinate space -The coordinate space defined by applying the internal transform of a frame to a point in *frame coordinate space* .

content element -A data item that can be seen by the user and is presented by a part's content model. Content elements can be manipulated through the graphical or scripting interface to a part.

content extent -The vertical dimension of the content area of a part in a frame. Content extent is used to calculate *bias transforms* .

content model -The specification of a part's contents (the data types of its content elements) and its content operations (the actions that can be performed on it and the interactions among its content elements).

content object -A content element that can be represented as an object and thus accessed and manipulated through semantic events.

content operation -A user action that manipulates a content element.

content property -A visual or behavioral characteristic of a containing part, such as its text font, that it makes available for embedded parts to adopt. Embedded parts can adopt the content properties of their containing parts, thus giving a more uniform appearance to a set of parts. Contrast with *property* and *Info property* .

content storage unit -The main storage unit of the Clipboard, drag-and-drop object, link source object, or link object.

content transform -The composite transform that converts from a part's content coordinates to its canvas coordinates.

content view type -See *frame view type* .

context -The outermost object in the object hierarchy defined by the direct parameter of an OSA event.

contextual help -In SAA Common User Access Architecture, help that gives specific information about the item the cursor is on. The help is contextual because it provides information about a specific item as it is currently being used. Contrast with *extended help* .

contiguous -Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control -In SAA Advanced Common User Access architecture, a component of the user interface that allows a user to select choices or type information; for example, a check box, an entry field, a radio button.

control area -A storage area used by a computer program to hold control information. (I) (A)

Control Program -(1) The basic functions of the operating system, including DOS emulation and the support for keyboard, mouse, and video input/output. (2) A computer program designed to schedule and to supervise the execution of programs of a computer system. (I) (A)

control window -A window that is used as part of a composite window to perform simple input and output tasks. Radio buttons and check boxes are examples.

control word -An instruction within a document that identifies its parts or indicates how to format the document.

conventional application -An application that directly handles events, opens documents, and is wholly responsible for manipulating, storing, and retrieving all of the data in its documents. Contrast with *application component* .

coordinate bias -The difference between a given coordinate system and *platform-normal coordinates* . Coordinate bias typically involves both a change in axis polarity and an offset.

coordinate space -A two-dimensional set of points used to generate output on a video display or printer.

Copy -A choice that places onto the clipboard, a copy of what the user has selected. See also *Cut* and *Paste* .

CORBA -See *Common Object Request Broker Architecture* .

core OSA event -An OSA event defined as part of the Core suite of OSA events in the *OSA Event Registry: Standard Suites* .

Core suite -The set of OSA events that any scriptable part is expected to support.

correlation -The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

coverage window -A window in which the application's help information is displayed.

CPI -Common Programming Interface.

critical extended attribute -An extended attribute that is necessary for the correct operation of the system or a particular application.

critical section -(1) In programming languages, a part of an asynchronous procedure that cannot be executed simultaneously with a certain part of another asynchronous procedure.

Note: Part of the other asynchronous procedure also is a critical section. (2) A section of code that is not reentrant; that is, code that can be executed by only one thread at a time.

CUA architecture -Common User Access architecture.

current draft -The most recent draft of an OpenDoc document. Only the current draft can be edited.

current frame -During drawing, the frame that is being drawn or within which editing is occurring.

current position -In computer graphics, the position, in user coordinates, that becomes the starting point for the next graphics routine, if that routine does not explicitly specify a starting point.

cursor -A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus* .

custom OSA event -An OSA event you define for use by your own applications. Instead of creating custom OSA events, you should try to use the standard OSA events and extend their definitions as necessary for your application. If you think you need to define custom OSA events, you should check with the OSA event Registrar to find out whether OSA events that already exist or are under development can be adapted to the needs of your application.

customizable -A level of scripting support of a part. A customizable part defines content objects and operations for interface elements such as menus and buttons; it allows the user to change its behavior during virtually any user action. Contrast with *scriptable* and *recordable* .

Cut -In SAA Common User Access architecture, a choice that removes a selected object, or a part of an object, to the clipboard, usually compressing the space it occupied in a window. See also *Copy* and *Paste* .

Glossary - D

daisy chain -A method of device interconnection for determining interrupt priority by connecting the interrupt sources serially.

database extension -The interface between the Data Access Manager and a data server.

data segment -A nonexecutable section of a program module; that is, a section of a program that contains data definitions.

data server -An application that acts as an interface between a database extension on a personal computer and a data source, which can be on the personal computer or on a remote host computer. A data server can be a database server program that can provide an interface to a variety of different databases, or it can be the data source itself, such as an application program.

data structure -The syntactic structure of symbolic expressions and their storage-allocation characteristics. (T)

data transfer -The movement of data from one object to another by way of the clipboard or by direct manipulation.

DBCS -Double-byte character set.

DDE -Dynamic data exchange.

deadlock -(1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug -To detect, diagnose, and eliminate errors in programs. (T)

decipoint -In printing, one tenth of a point. There are 72 points in an inch.

default container -The outermost container in an application's container hierarchy; usually the application itself. See also *container* hierarchy.

default editor for kind -A user-specified choice of part editor to use with parts whose *preferred editor* is not present.

default object accessor -Object accessors provided by OpenDoc that can be used to resolve content objects or properties of parts that do not themselves support scripting. Default accessors can return tokens representing an embedded frame, a standard Info property of a part, or a context switch (swap token).

default procedure -A function provided by the Presentation Manager Interface that may be used to process standard messages from dialogs or windows.

default scripting component -The scripting component used by the generic scripting component when an application passes kOSANullScript rather than a valid script ID to OSACompile or OSAScriptRecording.

default value -A value assumed when no value has been specified. Synonymous with assumed value. For example, in the graphics programming interface, the default line-type is 'solid'.

definition list -A type of list that pairs a term and its description.

delta -An application-defined threshold, or number of container items, from either end of the list.

derived class -See *subclass* .

descendant -See *child process* and *subclass* .

descriptive text -Text used in addition to a field prompt to give more information about a field.

descriptor list -A descriptor record of data type AEDescList whose data handle refers to a list of descriptor records.

descriptor record -A data structure of type AEDesc that consists of a handle to data and a descriptor type that identifies the type of the data referred to by the handle. Descriptor records are the fundamental data structures from which OSA events are constructed.

descriptor type -An identifier for the type of data referred to by the handle in a descriptor record.

Deselect all -A choice that cancels the selection of all of the objects that have been selected in that window.

desktop window -The window, corresponding to the physical device, against which all other types of windows are established.

destination content -The content at the destination of a link. It is a copy of the *source content* .

destination part -For a link, the part that displays the information copied from the source of the link. Contrast with *source part* .

detached process -A background process that runs independent of the parent process.

detent -A point on a slider that represents an exact value to which a user can move the slider arm.

device context -A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context*, and *screen device context*.

device driver -A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space -(1) Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units. (2) In computer graphics, a space defined by the complete set of addressable points of a display device. (A)

dialog -The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box -In SAA Advanced Common User Access architecture, a movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request. See also *message box*, *primary window*, *secondary window*. Also known as a *pop-up window*.

Dialog Box Editor -A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item -A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog procedure -A dialog window that is controlled by a window procedure. It is responsible for responding to all messages sent to the dialog window.

dialog tag language -A markup language used by the DTL compiler to create dialog objects.

dialog template -The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context -A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation -The user's ability to interact with an object by using the mouse, typically by dragging an object around on the Desktop and dropping it on other objects.

direct memory access (DMA) -A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit. (T)

direct parameter -The parameter in an OSA event that contains the data or object specifier record to be used by the server application. For example, a list of documents to be opened is specified in the direct parameter of the Open Documents event. See also *OSA event parameter*.

directory -A type of file containing the names and controlling information for other files or other directories.

dispatcher -The OpenDoc object that directs user events and semantic events to the correct part.

dispatch module -An OpenDoc object used by the dispatcher to dispatch events of a certain type to part editors.

display frame -A frame in which a part is displayed. A part's display frames are created by and embedded in its containing part. Contrast with *embedded frame*.

display-frames list -A part's list of all the frames in which it is displayed. If a part is displayed in only one frame, it has only one element in this list.

display point -Synonym for *pel*.

display property -A visual characteristic of a containing part, such as its text font, that it makes available for embedded parts to adopt. Embedded parts can adopt the display characteristics of their containing parts that they understand, thus giving a more uniform appearance to a set of parts. Display properties are stored as properties in a storage unit passed from containing part to embedded part.

Distributed SOM (DSOM) -Distributed System Object Model. A version of SOM that provides remote access to SOM objects in a transparent way that insulates client programmers from having to know the location or platform type where a target object will be instantiated. DSOM allows programmers to use the same object model independently of whether the objects they access are in the same process, in another process on the same machine, or across distributed networks.

dithering -(1) The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work, for example, on narrow lines. (2) In computer graphics, a technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

DMA -Direct memory access.

document -In OpenDoc, a user-organized collection of parts, all stored together.

document part -See *part*.

document process -A thread of execution that runs the document shell program. The document process provides the interface between the operating system and part editors. It accepts events from the operating system, provides the address space into which parts are loaded, and provides access to the window system and other features.

document shell -A program that provides an environment for all the parts in a document. The shell maintains the major document global databases: storage, window state, arbitrator, and dispatcher. This code also provides basic document behavior such as document creation, opening, saving, printing, and closing. OpenDoc provides a default document shell for each platform.

document window -A window that displays an OpenDoc document. The edges of the content area of the window represent the frame border of the document's root part. The OpenDoc document shell manages opening and closing of document windows. Contrast with *part window*.

DOS Protect Mode Interface (DPMI) -An interface between protect mode and real mode programs.

double-byte character set (DBCS) -A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword -A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit. (A)

DPMI -DOS Protect Mode Interface.

draft -A configuration of a document, defined at a certain point in time by the user. A document is made up of a set of drafts.

draft key -A number that identifies a specific cloning transaction.

draft permissions -A specification of the class of read/write access that a part editor has to a draft.

drag -In SAA Common User Access, to use a pointing device to move an object; for example, clicking on a window border, and dragging it to make the window larger.

drag and drop -A facility of OpenDoc that allows users to move or copy data through direct manipulation.

drag-copy -A drag-and-drop operation in which the dragged data remains at the source, and a copy is inserted at the destination.

drag-move -A drag-and-drop operation in which the dragged data is deleted from the source and inserted at the destination.

dragging -(1) In computer graphics, moving an object on the display screen as if it were attached to the pointer. (2) In computer graphics, moving one or more segments on a display surface by translating. (I) (A)

drawing canvas -The platform-specific drawing environment on which frames are laid out. Each window or printing device has one drawing canvas. See also *canvas*, *static canvas*, and *dynamic canvas*.

drawing chain -See *segment chain*.

drop -To fix the position of an object that is being dragged, by releasing the select button of the pointing device. See also *drag*.

DSOM -Distributed System Object Model. A version of SOM that works transparently over a network.

DTL -Dialog tag language.

DTS -Direct-To-SOM.

dual-boot function -A feature of the OS/2 operating system that allows the user to start DOS from within the operating system, or an OS/2 session from within DOS.

duplex -Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic canvas -A drawing canvas that can potentially be changed, such as a window, that can be scrolled or paged to display different portions of a part's data. Contrast with *static canvas*.

dynamic data exchange (DDE) -A message protocol used to communicate between applications that share data. The protocol uses shared memory as the means of exchanging data between applications.

dynamic data formatting -A formatting procedure that enables you to incorporate text, bit maps or metafiles in an IPF window at execution time.

dynamic link library -A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking -The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic segments -Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another

without affecting the rest of the displayed picture.

dynamic storage -(1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (A) (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (A) (3) See also *static storage* .

dynamic time slicing -Varies the size of the time slice depending on system load and paging activity.

dynamic-link module -A module that is linked at load time or run time.

Glossary - E

EBCDIC -Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

edge-triggered -Pertaining to an event semaphore that is posted then reset before a waiting thread gets a chance to run. The semaphore is considered to be posted for the rest of that thread's waiting period; the thread does not have to wait for the semaphore to be posted again.

edit-in-place -See *in-place editing* .

editor of last resort -The part editor that displays any part for which there is no available part editor on the system. The editor of last resort typically displays a gray rectangle representing the part's frame.

editor properties -A notebook, accessed through the **Edit** menu, in which the user can view and change properties for the part editor of the currently active part.

EGA -Extended graphics adapter.

element -(1) An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer. (2) An OSA event object contained by another OSA event object specified as the element's container. An OSA event object can contain many elements of the same element class, whereas an OSA event object can have only one of each of its properties. See also *OSA event object* , *container* , *element classes* , *property* .

element classes -In the *OSA Event Registry: Standard Suites* , a list of the object classes for the elements that an OSA event object of a given object class can contain. See also *OSA event object* , *object class* .

embed -To display one part in a frame within another part. The embedded part retains its identity as a separate part from the containing part. Contrast with *incorporate* .

embedded content -Content displayed in an embedded frame. A containing part editor does not directly manipulate embedded content. Contrast with *intrinsic content* .

embedded frame -A frame that displays an embedded part. The embedded frame itself is considered intrinsic content of the containing part; the part displayed within the frame is not.

embedded-frames list -A containing part's private list of all the frames embedded within it.

embedded part -A part displayed in an embedded frame. The data for an embedded part is stored within the same draft as its containing part. An embedded part is copied during a duplication of its containing part. An embedded part may itself be a containing part, unless it is a *noncontainer part* .

embedding part -A part that is capable of embedding other parts within its content; that is, it is capable of being a containing part. See also *container part* . Contrast with *nonembedding part* .

EMS -Expanded Memory Specification.

encapsulation -Hiding an object's implementation, that is, its private, internal data and methods. Private variables and methods are accessible only to the object that contains them.

entry field -In SAA Common User Access architecture, an area where a user types information. Its boundaries are usually indicated. See also *selection field* .

entry-field control -The component of a user interface that provides the means by which the application receives data entered by the user in an entry field. When it has the input focus, the entry field displays a flashing pointer at the position where the next typed character will go.

entry panel -A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory

text.

environment parameter -A parameter used by all methods of SOM objects to pass exceptions.

environment segment -The list of environment variables and their values for a process.

environment strings -ASCII text strings that define the value of environment variables.

environment variables -Variables that describe the execution environment of a process. These variables are named by the operating system or by the application. Environment variables named by the operating system are PATH, DPATH, INCLUDE, INIT, LIB, PROMPT, and TEMP. The values of environment variables are defined by the user in the CONFIG.SYS file, or by using the SET command at the OS/2 command prompt.

error callback function -An object callback function that gives the OSA Event Manager an address. The OSA Event Manager writes to this address the descriptor record it is currently working with if an error occurs during the resolution of an object specifier record. See also *object callback function* .

error message -An indication that an error has been detected. (A)

event -See *user event* . Contrast with *semantic event* .

event class -An attribute that identifies a group of related OSA events. The event class appears in the message field of the OSA event's event record. The event class and the event ID identify the action an OSA event performs. See also *OSA event attribute* , *event ID* .

event handler -(1) A routine that executes in response to receiving a user event. (2) See *semantic-event handler* .

event-info structure -A data structure that carries information about an OpenDoc user event in addition to that provided by the event structure.

event ID -An attribute that identifies a particular OSA event within a group of related OSA events. The event ID appears in the where field of the OSA event's event record. The event ID and the event class identify the action an OSA event performs. See also *OSA event attribute* , *event class* .

Event Manager -The collection of routines that an application can use to receive information about actions performed by the user, to receive notice of changes in the processing status of the application, and to communicate with other applications.

event semaphore -A semaphore that enables a thread to signal a waiting thread or threads that an event has occurred or that a task has been completed. The waiting threads can then perform an action that is dependent on the completion of the signaled event.

event structure -A platform-specific structure that carries information about an OpenDoc user event.

exception -In programming languages, an abnormal situation that may arise during execution, that may cause a deviation from the normal execution sequence, and for which facilities exist in a programming language to define, raise, recognize, ignore, and handle it.

exclusive focus -A focus that can be owned by only one frame at a time. The selection focus, for example, is exclusive; the user can edit within only one frame at a time. Contrast with *non-exclusive focus* .

exclusive system semaphore -A system semaphore that can be modified only by threads within the same process.

executable file -(1) A file that contains programs or commands that perform operations or actions to be taken. (2) A collection of related data records that execute programs.

exit -To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel* .

expanded memory specification (EMS) -Enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute -An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended-choice selection -A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection* .

extended help -In SAA Common User Access architecture, a help action that provides information about the contents of the application window from which a user requested help. Contrast with *contextual help* .

extension -An OpenDoc object that extends the programming interface of another OpenDoc object. Part editors, for example, can provide additional interfaces through extensions. An object class that duplicates all the characteristics of an object class of the same name and adds some of its own. Like a word in a dictionary, a single object class ID can have several related definitions.

extent -Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

external link -In Information Presentation Facility, a link that connects external online document files.

external transform -A transform that is applied to a facet to position, scale, or otherwise transform the facet and the image drawn within it. The external transform locates the facet in the coordinate space of the frame's containing part. Contrast with *internal transform* .

externalize -For a part or other OpenDoc object, to transform its in-memory representation into a persistent form in a storage unit. See also *write* . Contrast with *internalize* .

extracted draft -A draft that is extracted from a document into a new document.

Glossary - F

facet -An object that describes where a frame is displayed on a canvas.

factoring -Using OSA events to separate the code that controls an application's user interface from the code that responds to the user's manipulation of the interface. In a fully factored application, any significant user actions generate OSA events that a scripting component can record as statements in a compiled script. See also *recordable application* .

factory method -A method in one class that creates an instance of another class.

family-mode application -An application program that can run in the OS/2 environment and in the DOS environment; however, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT -File allocation table.

FEA -Full extended attribute.

fidelity -The faithfulness of translation attained (or attainable) between data of different part kinds. For a given part kind, other part kinds are ranked in fidelity by the level at which their editors can translate its data without loss.

field-level help -Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

FIFO -First-in-first-out. (A)

file -A named set of records stored or processed as a unit. (T)

file allocation table (FAT) -In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute -Any of the attributes that describe the characteristics of a file.

file specification -The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system -The combination of software and hardware that supports storing information on a storage device.

file system driver (FSD) -A program that manages file I/O and controls the format of information on the storage media.

fillet -A curve that is tangential to the end points of two adjoining lines. See also *polyfillet* .

filtering -An application process that changes the order of data in a queue.

first-in-first-out (FIFO) -A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag -(1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) A characteristic of a file or directory that enables it to be used in certain ways. See also *archive flag* , *hidden flag* , and *read-only flag* .

focus -A designation of ownership of a shared resource such as menus, selection, keystrokes, and serial ports. The part that owns a focus has use of that shared resource.

focus module -An OpenDoc object used by the arbitrator to assign an owner or owners to a given focus type.

focus set -A group of foci requested as a unit.

folder -A container used to organize objects.

font -A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

Font Editor -A utility program provided with the IBM Developers Toolkit that enables the design and creation of new fonts.

foreground program -(1) The program with which the user is currently interacting. Also known as *interactive program* . Contrast with *background program* . (2) In multiprocessing, a high-priority program.

frame -(1) The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. (2) In OpenDoc, a bounded portion of the content area of a part, defining the location of an embedded part. The edge of a frame marks the boundary between intrinsic content and embedded content. A frame can be a rectangle or any other, even irregular, shape.

frame border -A visual indication of the boundary of a frame. The appearance of the frame border indicates the state of the frame (active, inactive, or selected). The frame border is drawn and manipulated by the containing part or by OpenDoc, not by the part within the frame.

frame coordinate space -The coordinate space in which a part's frame shape, used shape, active shape, and clip shape are defined. Contrast with *content coordinate space* . See also *window coordinate space* , *canvas coordinate space* .

frame group -A set of embedded frames that a containing part designates as related, for purposes such as flowing content from one frame to another. Each frame group has its own *group ID* ; frames within a frame group have a *frame sequence* .

frame negotiation -The process of adjusting the size and shape of an embedded frame. Embedded parts can request changes to their frames, but the containing parts control the changes that occur.

frame sequence -The order of frames in a frame group.

frame shape -A shape that defines a frame and its border, expressed in terms of the frame's local coordinate space.

frame styles -Standard window layouts provided by the Presentation Manager.

frame transform -The composite transform that converts from a part's frame coordinates to its canvas coordinates

frame view type -A view type in which all or a portion of a part's contents is displayed within a frame, the border of which is visible when the part is active or selected. Other possible view types for displaying a part include large icon, small icon, and thumbnail. Frame view type is sometimes called content view type.

FSD -File system driver.

full-duplex -Synonym for *duplex* .

full-screen application -An application that has complete control of the screen.

fully scriptable -Characteristic of a scriptable part in which semantic events can invoke any action a user might be able to perform.

function -(1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key -A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area -The area at the bottom of a window that contains function key assignments such as F1=Help.

functional-area OSA event -A standard OSA event supported by applications with related features; for example, an OSA event related to text manipulation for word-processing applications, or an OSA event related to graphics manipulation for drawing applications. Functional-area OSA events are defined by CI Labs, in consultation with interested developers and are published in the *OSA Event Registry: Standard Suites* .

Glossary - G

GDT -Global Descriptor Table.

general protection fault -An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPB code segment* .

Global Descriptor Table (GDT) -A table that defines code and data segments available to all tasks in an application.

global dynamic-link module -A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character -Either a question mark (?) or an asterisk (*) used as a variable in a file name or file name extension when referring to a particular file or group of files.

glyph -A graphic symbol whose appearance conveys information.

GPI -Graphics programming interface.

graphic primitive -In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures. See also *graphics segment* .

graphics -(1) A picture defined in terms of graphic primitives and graphics attributes. (2) The making of charts and pictures. (3) Pertaining to charts, tables, and their creation. (4) See *computer graphics, coordinate graphics, fixed-image graphics, interactive graphics, passive graphics, raster graphics* .

graphics attributes -Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes* .

graphics field -The clipping boundary that defines the visible part of the presentation-page contents.

graphics mode -One of several states of a display. The mode determines the resolution and color content of the screen.

graphics model space -The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space* .

Graphics programming interface -The formally defined programming language that is between an IBM graphics program and the user of the program.

graphics segment -A sequence of related graphic primitives and graphics attributes. See also *graphic primitive* .

graphics system -A specific drawing architecture. Some graphics systems (such as Display PostScript) are available on more than one platform; some platforms support more than one graphics system.

graying -The indication that a choice on a pull-down is unavailable.

group -A collection of logically connected controls. For example, the buttons controlling paper size for a printer could be called a group. See also *program group* .

group ID -In OpenDoc, a number that identifies a frame group, assigned by the group's containing part.

Glossary - H

handle -(1) An identifier that represents an object, such as a device or window, to the Presentation Interface. (2) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

hard error -An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

header -(1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

heading tags -A document element that enables information to be displayed in windows, and that controls entries in the contents window controls placement of push buttons in a window, and defines the shape and size of windows.

heap -An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

help function -(1) A function that provides information about a specific field, an application panel, or information about the help facility. (2) One or more display images that describe how to use application software or how to do a system operation.

Help index -In SAA Common User Access architecture, a help action that provides an index of the help information available for an application.

help panel -A panel with information to assist users that is displayed in response to a help request from the user.

help window -A Common-User-Access-defined secondary window that displays information when the user requests help.

hide button -In the OS/2 operating system, a small, square button located in the right-hand corner of the title bar of a window that, when selected, removes from the screen all the windows associated with that window. Contrast with *maximize button* . See also *restore button* .

hidden file -An operating system file that is not displayed by a directory listing.

hierarchical inheritance -The relationship between parent and child classes. An object that is lower in the inheritance hierarchy than another object, inherits all the characteristics and behaviors of the objects above it in the hierarchy.

hierarchy -A tree of segments beginning with the root segment and proceeding downward to dependent segment types.

high-performance file system (HPFS) -In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters.

hit testing -The means of identifying which window is associated with which input device event.

hook -A point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function.

hook chain -A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot part -A part, such as a control, that performs an action (like running a script), rather than activating itself, when it receives a mouse click.

hot spot -The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

HPFS -high-performance file system.

hypergraphic link -A connection between one piece of information and another through the use of graphics.

hypertext -A way of presenting information online with connections between one piece of information and another, called *hypertext links*. See also *hypertext link*.

hypertext link -A connection between one piece of information and another.

Glossary - I

I/O operation -An input operation to, or output operation from a device attached to a computer.

I-beam pointer -A pointer that indicates an area, such as an entry field in which text can be edited.

icon -(1) In SAA Advanced Common User Access architecture, a graphical representation of an object, consisting of an image, image background, and a label. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs. (2) In OpenDoc, a small, type-specific picture with a name. Possible iconic view types for displaying a part include as a (standard) *large icon*, *small icon*, or *thumbnail*; the other possible view type is in a *frame*.

icon area -In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor -The Presentation Manager-provided tool for creating icons.

identity transform -A transform that has no effect on points to which it is applied.

IDL -Interface Definition Language.

image font -A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of one of the symbols. Contrast with *outline font*.

implementation binding -See *private header file*.

implied length -The definition of a specific length for a data type. An example of this is the Data Access Manager's typeInteger data type, which has a defined length of 4 bytes.

implied metaclass -Subclassing the metaclass of a parent class without a separate IDL for the resultant metaclass.

inactive frame -A frame that does not have the selection focus.

inactive part -A part that has no active display frames.

incorporate -To merge the data from one part into the contents of another part so that the merged data retains no separate identity as a part. Contrast with *embed* .

indirect manipulation -Interaction with an object through choices and controls.

information device context -A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context* .

information panel -A defined panel type characterized by a body containing only protected information.

Information Presentation Facility (IPF) -A facility provided by the OS/2 operating system, by which application developers can produce online documentation and context-sensitive online help panels for their applications.

inheritance -The passing of class resources or attributes from a parent class downstream in the class hierarchy to a child class. The new class inherits all the data and methods of the parent class without having to redefine them.

in-place editing -User manipulation of data in an embedded part without leaving the context of the document in which the part is displayed (for example, without opening a new window for the part).

input focus -(1) The area of a window where user interaction is possible using an input device, such as a mouse or the keyboard. (2) The position in the *active window* where a user's normal interaction with the keyboard will appear.

input router -An internal OS/2 process that removes messages from the system queue.

input/output control -A device-specific command that requests a function of a device driver.

insertion location descriptor record -A record of type `typeInsertionLoc` that consists of two keyword-specified descriptor records. The first is an object specifier record, and the data for the second is a constant that specifies the insertion location in relation to the OSA event object described by the object specifier record.

inside-out activation -A mode of user interaction in which a mouse click anywhere in a document activates the smallest possible enclosing frame and performs the appropriate selection action on the content element at the click location. OpenDoc uses inside-out selection. Contrast with *outside-in activation* .

inside-out selection -A mode of user interaction in which a mouse click anywhere in a document activates the smallest possible enclosing frame and performs the appropriate selection action on the content element at the click location. OpenDoc uses inside-out selection. Contrast with *outside-in selection* .

installable file system (IFS) -A file system in which software is installed when the operating system is started.

instance -A single occurrence of an object class that has a particular behavior. See also *object* .

instantiate -(1) To make an instance of; to replicate. (2) In object-oriented programming, to represent a class abstraction with a concrete instance of the class.

instruction pointer -A pointer that provides addressability for a machine interface instruction in a program.

integer atom -An *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics -Graphics that can be moved or manipulated by a user at a terminal.

interactive program -(1) A program that is running (active) and is ready to receive (or is receiving) input from a user. (2) A running program that can receive input from the keyboard or another input device. Contrast with *active program* and *noninteractive program* .

Also known as a *foreground program* .

interapplication communication (IAC) architecture -A standard and extensible mechanism for communication among applications, including the Open Scripting Architecture, the OSA Event Manager, and the Event Manager.

interchange file -A file containing data that can be sent from one application to another.

Interface Definition Language (IDL) -Language-neutral syntax created by IBM to describe the interface of classes that can be compiled by the SOM compiler.

internal transform -A transform that positions, scales, or otherwise transforms the image of a part drawn within a frame. Contrast with *external transform* .

internalize -For a part or other OpenDoc object, to transform its persistent form in a storage unit into an appropriate in-memory representation. Contrast with *externalize* . See also *read* .

interoperability -Access to an OpenDoc part or document from different platforms or with different software systems.

interpreter -A program that translates and executes each instruction of a high-level programming language before it translates and executes.

interprocess communication (IPC) -In the OS/2 operating system, the exchange of information between processes or threads through semaphores, pipes, queues, and shared memory.

interval timer -(1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

intrinsic content -The content elements native to a particular part, as opposed to other parts embedded in it. Contrast with *embedded content* .

invalid shape -The area of a frame, facet, or canvas that needs redrawing. Update events cause redrawing of the invalid area.

invalidate -To mark an area of a canvas (or facet, or frame) as in need of redrawing.

invariant -An aspect of the internal state of an object that must be maintained for the object to behave properly according to its design.

IOCtl -Input/output control.

IOPL -Input/output privilege level.

IOPL code segment -An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level* .

IPC -Interprocess communication.

IPF -Information Presentation Facility.

IPF compiler -A text compiler that interpret tags in a source file and converts the information into the specified format.

IPF tag language -A markup language that provides the instructions for displaying online information.

ISO string -A null-terminated 7-bit ASCII string.

item -A data object that can be passed in a DDE transaction.

iterator -A class or object that provides sequential access to a collection of objects of another class. A part's embedded-frames iterator, for example, provides access to all of the part's embedded frames.

Glossary - J

journal -A special-purpose file that is used to record changes made in the system.

Glossary - K

Kanji -A graphic character set used in Japanese ideographic alphabets.

KBD\$ -Character-device name reserved for the keyboard.

kernel -The part of an operating system that performs basic functions, such as allocating hardware resources.

Kerning -The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

key data -The data in an object specifier record that distinguishes one or more OSA event objects from other OSA event objects of the same object class in the same container. Key data is specified by a keyword-specified descriptor record with the keyword keyAEKeyData. The OSA Event Manager interprets key data according to the key form specified in the same object specifier record.

key form -The form taken by the key data in an object specifier record. The key form is specified by a keyword-specified descriptor record with the keyword keyAEKeyForm. The keyword-specified descriptor record contains a constant that determines how the OSA Event Manager and a target application use the key data to locate specific OSA event objects. For example, the key form constant formName indicates that the key data consists of a name, which should be compared to the names of OSA event objects in the container specified by the object specifier record.

keyboard accelerator -A keystroke that generates a command message for an application.

keyboard augmentation -A function that enables a user to press a keyboard key while pressing a mouse button.

keyboard focus -A temporary attribute of a window. The window that has a keyboard focus receives all keyboard input until the focus changes to a different window.

Keys help -In SAA Common User Access architecture, a help action that provides a listing of the application keys and their assigned functions.

keystroke focus -A designation of ownership of keystroke events. The part whose frame has the keystroke focus receives keystroke events.
See also *selection focus* .

keystroke focus frame -The frame to which keystroke events are to be sent.

keyword -A four-character code that uniquely identifies a descriptor record inside another descriptor record. In OSA Event Manager functions, constants are typically used to represent the four-character codes.

keyword-specified descriptor record -A record of data type AEKeyDesc that consists of a keyword and a descriptor record.
Keyword-specified descriptor records are used to describe the attributes and parameters of an OSA event.

kind -See *part kind* .

Glossary - L

label -In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LAN -local area network.

language support procedure -A function provided by the Presentation Manager Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

large icon view type -A view type in which a part is represented by a 32 by 32-pixel bitmap image. Other possible view types for displaying a part include small icon, thumbnail, and frame.

layout -The process of arranging frames and content elements in a document for drawing.

lazy drag -See *pickup and drop* .

lazy drag set -See *pickup set* .

lazy instantiation -The process of creating objects (such as embedded frames) in memory only when they are needed for display, such as when the user scrolls them into view. Lazy instantiation can help minimize the memory requirements of your parts.

LDT -In the OS/2 operating system, Local Descriptor Table.

leaf part -See *noncontainer part* .

LIFO stack -A stack from which data is retrieved in last-in, first-out order.

linear address -A unique value that identifies the memory object.

link -(1) A persistent reference to a part or to a set of content elements of a part. (2) An OpenDoc object that represents a link destination.

link destination -The portion of a part's content area that represents the destination of a link.

link key -A number that identifies a specific transaction to access a link object or link-source object.

link manager -An OpenDoc object that coordinates cross-document links.

link source -The portion of a part's content area that represents the source of a link.

link specification -An object, placed on the clipboard or in a drag-and-drop object, from which the source part (the part that placed the data) can construct a link if necessary.

link status -The link-related state (in a link source, in a link destination, or not in a link) of a frame.

linked list -Synonym for *chained list* .

linked part -A part (or a portion of a part's content data) that appears to the user to be embedded in one part, but it is actually embedded in a different part. Linked data is not copied when the link's containing part is duplicated; a new link is created instead.

list box -In SAA Advanced Common User Access architecture, a control that contains scrollable choices from which a user can select one choice.

Note: In CUA architecture, this is a programmer term. The end user term is selection list.

list button -A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

list panel -A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load -For a part editor, to transform the persistent form of a part in a draft into an appropriate in-memory representation, which can be a representation of the complete part or only a subset, depending on the current display requirements of the document. Contrast with *save*.

load-on-call -A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

load time -The point in time at which a program module is loaded into main storage for execution.

local area network (LAN) -(1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

Note: A LAN does not use store and forward techniques. (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

Local Descriptor Table (LDT) -Defines code and data segments specific to a single task.

lock -A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

logical descriptor record -A coerced AE record of type `typeLogicalDescriptor` that specifies a logical expression-that is, an expression that the OSA Event Manager evaluates to either TRUE or FALSE. The logical expression is constructed from a logical operator (one of the Boolean operators AND, OR, or NOT) and a list of logical terms to which the operator is applied. Each logical term in the list can be either another logical descriptor record or a comparison descriptor record.

logical storage device -A device that the user can map to a physical (actual) device.

LPT1, LPT2, LPT3 -Character-device names reserved for parallel printers 1 through 3.

Glossary - M

main storage unit -The storage unit that holds the contents property (`kODPropContents`) of a part. A part's main storage unit, plus possibly other auxiliary storage units referenced from it, holds all of a part's content.

main window -The window that is positioned relative to the *desktop window*.

manipulation button -The button on a pointing device a user presses to directly manipulate an object.

map -(1) A set of values having a defined correspondence with the quantities or values of another set. (I) (A) (2) To establish a set of values having a defined correspondence with the quantities or values of another set. (I)

mark-adjusting function -A marking callback function that unmarks objects previously marked by a call to an application's marking function.

mark count -The number of times the OSA Event Manager has called the marking function for the current mark token. Applications that support marking callback functions should associate the mark count with each OSA event object they mark.

mark token -A token returned by a mark token function. A mark token identifies the way an application marks OSA event objects during the current sessions while resolving a single test. A mark token does not identify a specific OSA event object; rather, it allows an application that supports marking callback functions to associate a group of objects with a marked set.

mark token function -A marking callback function that returns a mark token.

marker box -In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol -A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

marking callback functions -Object callback functions that allow your application to use its own marking scheme rather than tokens when identifying large groups of OSA event objects. See also *mark-adjusting function* , *mark token function* , *object callback function* , and *object-marking function* .

marquee box -The rectangle that appears during a selection technique in which a user selects objects by drawing a box around them with a pointing device.

Master Help Index -In the OS/2 operating system, an alphabetic list of help topics related to using the operating system.

maximize -To enlarge a window to its largest possible size.

media window -The part of the physical device (display, printer, or plotter) on which a picture is presented.

member function -See *method* .

memory block -Part memory within a heap.

memory device context -A logical description of a data destination that is a memory bit map. See also *device context* .

memory management -A feature of the operating system for allocating, sharing, and freeing main storage.

memory object -Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint.

menu -In SAA Advanced Common User Access architecture, an extension of the menu bar that displays a list of choices available for a selected choice in the menu bar. After a user selects a choice in menu bar, the corresponding menu appears. Additional pop-up windows can appear from menu choices.

menu bar -In SAA Advanced Common User Access architecture, the area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

menu button -The button on a pointing device that a user presses to view a pop-up menu associated with an object.

message -(1) In the Presentation Manager, a packet of data used for communication between the Presentation Manager interface and Presentation Manager applications (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process. See also *semantic event* .

message block -A byte stream that an open application uses to send data to and receive data from another open application (which can be located on the same computer or across a network).

message box -(1) A dialog window predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. (2) In SAA Advanced Common User Access architecture, a type of window that shows messages to users. See also *dialog box* , *primary window* , *secondary window* .

message filter -The means of selecting which messages from a specific window will be handled by the application.

message interface -An OpenDoc object that provides an interface to allow parts to send messages (semantic events) to other parts, either in the same document or in other documents.

message queue -A sequenced collection of messages to be read by the application.

message stream mode -A method of operation in which data is treated as a stream of messages. Contrast with *byte stream* .

metacharacter -See *global file-name character* .

metaclass -The conjunction of an object and its class information; that is, the information pertaining to the class as a whole, rather than to a single instance of the class. Each class is itself an object, which is an instance of the metaclass.

metafile -A file containing a series of attributes that set color, shape and size, usually of a picture or a drawing. Using a program that can interpret these attributes, a user can view the assembled image.

metafile device context -A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context* .

metalanguage -A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

method -A function that manipulates the data of a particular class of objects.

method override -The replacement, by a child class, of the implementation of a method inherited from a parent and an ancestor class.

mickey -A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space -A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize -To remove from the screen all windows associated with an application and replace them with an icon that represents the

application.

mix -An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix* . Contrast with *background mix* .

mixed character string -A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic -(1) A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item. (2) In SAA Advanced Common User Access architecture, usually a single character, within the text of a choice, identified by an underscore beneath the character. If all characters in a choice already serve as mnemonics for other choices, another character, placed in parentheses immediately following the choice, can be used. When a user types the mnemonic for a choice, the choice is either selected or the cursor is moved to that choice.

modal dialog box -In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that requires a user to enter information before continuing to work in the application window from which it was displayed. Contrast with *modeless dialog box* . Also known as a *serial dialog box* . Contrast with *parallel dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

modal focus -A designation of ownership of the right to display modal dialog boxes. A part displaying a modal dialog must first acquire the modal focus, so that other parts cannot do the same until the first part is finished.

model space -See *graphics model space* .

modeless dialog box -In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that allows users to continue their dialog with the application without entering information in the dialog box. Also known as a *parallel dialog box* . Contrast with *modal dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

module definition file -A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

monitor -A special use of a dispatch module, in which it is installed in order to be notified of events, but does not dispatch them.

monolithic application -See *conventional application* .

mouse -In SAA usage, a device that a user moves on a flat surface to position a pointer on the screen. It allows a user to select a choice or function to be performed or to perform operations on the screen, such as dragging or drawing lines from one position to another.

mouse region -An area (by default a size of 1 pixel square) within which the user can move the mouse pointer without triggering an event.

MOUSE\$ -Character-device name reserved for a mouse.

multiple-choice selection -In SAA Basic Common User Access architecture, a type of field from which a user can select one or more choices or select none. See also *check box* . Contrast with *extended-choice selection* .

multiple-line entry field -In SAA Advanced Common User Access architecture, a control into which a user types more than one line of information. See also *single-line entry field* .

multitasking -The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex semaphore -(Mutual exclusion semaphore). A semaphore that enables threads to serialize their access to resources. Only the thread that currently owns the mutex semaphore can gain access to the resource, thus preventing one thread from interrupting operations being performed by another.

muxwait semaphore -(Multiple wait semaphore). A semaphore that enables a thread to wait either for multiple event semaphores to be posted or for multiple mutex semaphores to be released. Alternatively, a muxwait semaphore can be set to enable a thread to wait for any ONE of the event or mutex semaphores in the muxwait semaphore's list to be posted or released.

Glossary - N

name resolver -An OpenDoc object that determines the proper recipient of a semantic event. The name resolver can resolve *object* specifiers, permitting semantic events to be sent to individual objects within a part.

name space -An object consisting of a set of text strings used to identify kinds of objects or classes of behavior, for registration purposes. For example, OpenDoc uses name spaces to identify part kinds and categories for binding.

name-space manager -An OpenDoc object that creates and deletes *name spaces* .

named pipe -A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes.
Contrast with *unnamed pipe* .

national language support (NLS) -The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

nested list -A list that is contained within another list.

NLS -national language support.

non-8.3 file-name format -A file-naming convention in which file names can consist of up to 255 characters. See also *8.3 file-name format* .

non-exclusive focus -A focus that can be owned by more than one frame at a time. OpenDoc supports the use of nonexclusive foci. Contrast with *exclusive focus* .

noncontainer part -A part that cannot itself contain embedded parts. A noncontainer part can never be a *containing part* . Contrast with *container part* .

noncritical extended attribute -An extended attribute that is not necessary for the function of an application.

nondestructive read -Reading that does not erase the data in the source location. (T)

nonembedding part -see *noncontainer part* . Contrast with *embedding part* .

noninteractive program -A running program that cannot receive input from the keyboard or other input device. Contrast with *active program* and *interactive program* .

nonpersistent frame -A frame that exists as an object in memory, but has no storage unit and is not stored persistently.

nonretained graphics -Graphic primitives that are not remembered by the Presentation Manager interface when they have been drawn.
Contrast with *retained graphics* .

null character (NUL) -(1) Character-device name reserved for a nonexistent (dummy) device. (2) A control character that is used to accomplish media-fill or time-fill and that may be inserted into or removed from a sequence of characters without affecting the meaning of the sequence; however, the control of equipment or the format may be affected by this character. (I) (A)

null descriptor record -A descriptor record whose descriptor type is typeNull and whose data handle is NIL.

null-terminated string -A string of (n+1) characters where the (n+1)th character is the 'null' character (0x00) Also known as 'zero-terminated' string and 'ASCIIZ' string.

Glossary - O

object - A programming entity, existing in memory at run time, that is an individual instantiation of a particular *class* .

object accessor -A function called by the name resolver to resolve semantic-event object specifiers.

object accessor dispatch table -A table in shared memory that the OSA Event Manager uses to map descriptions of objects in an object specifier record to object accessor functions that can locate those objects.

object accessor function -An application-defined function that locates an OSA event object of a specified object class in a container identified by a token of a specified descriptor type.

object callback -A function called by the name resolver to allow your part to provide extra information needed for semantic-event object resolution.

object callback function -An application-defined function used by the OSA Event Manager to resolve object specifier records. See also *error callback function* , *marking callback functions* , *object-comparison function* , *object-counting function* , and *token disposal function* .

object class -A category for OSA event objects that share specific characteristics listed in an object class definition in the *OSA Event Registry: Standard Suites* . Among these characteristics are properties, element classes, and OSA events that can specify objects of that class. An object class is specified in an object specifier record by a keyword-specified descriptor record with the keyword keyAEDesiredClass whose data handle refers to either a constant or an object class ID.

object class ID -A four-character code, which can also be represented by a constant, that identifies an object class for an OSA event object. The object class ID for a primitive object class is the same as the four-character value of its descriptor type.

object class inheritance hierarchy -The hierarchy of subclasses and superclasses that determines which properties, elements, and OSA events object classes inherit from other object classes.

object-comparison function -An object callback function that compares an element to either another element or to a descriptor record and returns either TRUE or FALSE.

object-counting function -An object callback function that counts the number of elements of a specified class in a specified container, so that the OSA Event Manager can determine how many elements it must examine to find the element or elements that pass a test.

Object Interface Definition Language (OIDL) -Specification language used in SOM Version 1 for defining classes. Replaced by Interface Definition Language (IDL).

Object Linking and Embedding (OLE) -An application protocol developed by Microsoft Corporation that allows objects created by one application to be linked to or embedded in objects created by another application.

Object Management Group (OMG) -An industry consortium that promulgates standards for object programming.

object-marking function -An object callback function called repeatedly by the OSA Event Manager to mark specific OSA event objects. See also *marking callback functions* .

object model -A feature of OSA events that allows a part to define a hierarchical arrangement of content objects to represent the elements of the part's content.

Object REXX component -The scripting component that implements the Object REXX scripting language. See also *scripting component* .

Object REXX scripting language -The standard user scripting language defined by IBM. The Object REXX scripting language is implemented by the Object REXX scripting component.

object specifier -A designation of a content object within a part, used to determine the target of a semantic event. Object specifiers can be names ("blue rectangle") or logical designations ("word 1 of line 2 of embedded frame 3").

object specifier record -A description of one or more OSA event objects based on the OSA Event Manager and the classification system defined in the *OSA Event Registry: Standard Suites* . An object specifier record consists of a descriptor record of descriptor type `typeObjectSpecifier` that comprises four keyword-specified descriptor records: the object class ID, the container for the OSA event object (which is usually another OSA event object, specified by another object specifier record), the key form, and the key data.

object window -A window that does not have a parent but which might have child windows. An object window cannot be presented on a device.

OIDL -Object Interface Definition Language.

OLE -See *Object Linking and Embedding* .

OLE interoperability -A technology that enables seamless interoperability between OpenDoc and Microsoft Corporation's Object Linking and Embedding (OLE) technology for interapplication communication. It allows OLE objects to function automatically as parts in OpenDoc documents, and OpenDoc parts to function automatically as OLE objects in OLE containers.

open -To start working with a file, directory, or other object.

Open Application event -An OSA event that asks an application to perform the tasks-such as displaying untitled windows-associated with opening itself; one of the four required OSA events.

Open Documents event -An OSA event that asks an application to open one or more documents specified in a list; one of the four required OSA events.

Open Linking and Embedding of Objects (OLEO) -A technology that enables seamless interoperability between OpenDoc and Microsoft Corporation's Object Linking and Embedding (OLE) technology for interapplication communication. It allows OLE objects to function automatically as parts in OpenDoc documents, and OpenDoc parts to function automatically as OLE objects in OLE containers.

Open Scripting Architecture (OSA) -A mechanism based on the OSA Event Manager and the *OSA Event Registry: Standard Suites* that allows users to control multiple applications by means of scripts. The scripts can be written in any scripting language that supports the OSA.

OpenDoc -A multiplatform technology, implemented as a set of shared libraries, that uses component software to facilitate the construction and sharing of compound documents.

OpenDoc Development Framework (ODF) -A part-editor framework that facilitates creation of OpenDoc parts.

optional parameter -A supplemental parameter in an OSA event used to specify data that the server application can use in addition to the data specified in the direct parameter. Source applications list the keywords for parameters that they consider optional in the attribute identified by the `keyOptionalKeywordAttr` keyword. Target applications use this attribute to identify any parameters that they are required to understand. If a parameter's keyword is not listed in this attribute, the target application must understand that parameter to handle the event successfully. See also *OSA event attribute* , *OSA event parameter* .

ordered list -Vertical arrangements of items, with each item in the list preceded by a number or letter.

OSA event -An OSA event consists of attributes (including the event class and event ID, which identify the event and its task) and, usually, parameters (which contain data used by the target application for the event). See also *OSA event attribute* , *OSA event parameter* .

OSA event array -An array in a descriptor list. The data for an OSA event array is specified by an array data record, which is defined by the data type *AEArrayData*.

OSA event attribute -A keyword-specified descriptor record that identifies the event class, event ID, target application, or some other characteristic of an OSA event. Taken together, the attributes of an OSA event identify the event and denote the task to be performed on the data specified in the OSA event's parameters. Unlike OSA event parameters (which contain data used only by the target application of the OSA event), OSA event attributes contain information that can be used by both the OSA Event Manager and the target application. See also *OSA event parameter* .

OSA event dispatch table -A table in shared memory that the OSA Event Manager uses to map OSA events to the appropriate OSA event handlers.

OSA event handler -An application-defined function that extracts pertinent data from an OSA event, performs the action requested by the OSA event, and returns a result.

OSA Event Manager -The collection of routines that allows client applications to send OSA events to server applications for the purpose of requesting services or information.

OSA event object -A distinct item in a target application or any of its documents that can be specified by an object specifier record in an OSA event sent by a source application. OSA event objects can be anything that an application can locate on the basis of such a description, including items that a user can differentiate and manipulate while using an application, such as words, paragraphs, shapes, windows, or style formats. See also *object specifier record* .

OSA event object class -See *object class* .

OSA event parameter -A keyword-specified descriptor record containing data that the target application for an OSA event uses. Unlike OSA event attributes (which contain information that can be used by both the OSA Event Manager and the target application), OSA event parameters contain data used only by the target application of the OSA event. See also *OSA event attribute* , *direct parameter* , *optional parameter* , *required parameter* .

OSA event record -A descriptor record of data type *OSAEvent* that contains a list of keyword-specified descriptor records. These descriptor records describe-at least-the attributes necessary for an OSA event; they may also describe parameters for the OSA event. OSA Event Manager functions are used to add parameters to an OSA event record.

OSA event user terminology resources -Two resources with identical formats used by server applications to specify the OSA events and corresponding user terminology that the applications support. The 'aeut' resource, which is provided by scripting components, contains terminology information for all the standard suites of OSA events defined in the *OSA Event Registry: Standard Suites* . An 'aete' resource must be provided by every scriptable application; it describes which of the standard suites listed in the 'aeut' resource the application supports and provides additional terminology information for extensions to the standard suites and custom OSA events supported by the application. See also *scripting component* .

outline font -A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font* . Contrast with *image font* .

output area -An area of storage reserved for output. (A)

outside-in activation -A mode of user interaction in which a mouse click anywhere in a document activates the largest possible enclosing frame that is not already active. Contrast with *inside-out activation* .

outside-in selection -A mode of user interaction in which a mouse click anywhere in a document activates the largest possible enclosing frame that is not already active. Contrast with *inside-out selection* .

overlaid frame -An embedded frame that floats above the content (including other embedded frames) of its containing part, and thus need not engage in frame negotiation with the containing part.

override -To replace a method belonging to a superclass with a method of the same name in a subclass, in order to modify its behavior.

owner -For a canvas, the part that created the canvas and attached it to a facet. The owner is responsible for transferring the results of drawing on the canvas to its parent canvas.

owner window -A window into which specific events that occur in another (owned) window are reported.

ownership -The determination of how windows communicate using messages.

owning process -The process that owns the resources that might be shared with other processes.

Glossary - P

page -(1) A 4KB segment of contiguous physical memory. (2) A defined unit of space on a storage medium.

page viewport -A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint -(1) The action of drawing or redrawing the contents of a window. (2) In computer graphics, to shade an area of a display image; for example, with crosshatching or color.

panel -In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

panel area -An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel area separator -In SAA Basic Common User Access architecture, a solid, dashed, or blank line that provides a visual distinction between two adjacent areas of a panel.

panel body -The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body can contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area -See *client area* .

panel definition -A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID -In SAA Basic Common User Access architecture, a panel identifier, located in the upper-left corner of a panel. A user can choose whether to display the panel ID.

panel title -In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

paper size -The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box -See *modeless dialog box* .

parameter list -A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parent canvas -The canvas closest above a canvas in the facet hierarchy. If, for example, there is a single offscreen canvas attached to an embedded facet in a window, the window canvas (attached to the root facet) is the parent of the offscreen canvas.

parent class -See *superclass* .

parent process -In the OS/2 operating system, a process that creates other processes. Contrast with *child process* .

parent window -In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window also will be moved, resized, or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window. Contrast with *child window* .

part -A portion of a compound document; it consists of document content, plus-at run time-a part editor that manipulates that content. The content is data of a given structure or type, such as text, graphics, or video; the code is a part editor. In programming terms, a part is an object, an instantiation of a subclass of the class ODpart. To a user, a part is a single set of information displayed and manipulated in one or more frames or windows. Synonymous with *document part* .

part category -A general classification of the format of data handled by a part editor. Categories are broad classes of data format, meaningful to end-users, such as "text", "graphics", or "table". Contrast with *part kind* .

part container -See *container part* .

part content -The portion of a part that describes its data. In programming terms, the part content is represented by the instance variables of the part object; it is the state of the part and is the portion of it that is stored persistently. To the user, there is no distinction between part and part content; the user considers both the part content alone, and the content plus its part editor, as a part. Contrast with *part editor* and *part* . See also *intrinsic content* and *embedded content* .

part editor -An application component that can display and change the data of a part. It is the executable code that provides the behavior for the part. Contrast with *part content*, *part viewer* .

part ID -An identifier that uniquely names a part within the context of a document. This ID represents a storage unit ID within a particular draft of a document.

part info -(1) Part-specific data, of any type or size, used by a part editor to identify what should be displayed in a particular frame or facet and how it should be displayed. (2) Information about a given part that can be seen by the user and is displayed in the Part Info dialog box.

part kind -A specific classification of the format of data handled by a part editor. A kind specifies the specific data format handled by, and possibly native to, a part editor. Kinds are meaningful to end-users and have designations such as such as "MyEditor 2.0" or "MyEditor 1.0". Contrast with *part category* .

part property -A user-accessible characteristic of a part or a portion of its content. The user can modify some properties, such as the name of a part; the user cannot modify some other properties, such as its part category. See also *property* .

part registry -The mechanism by which the document shell maps parts to part editors according to their part kind.

part table -A list of all the parts contained within a document and a list of associated data.

part viewer -An application component that can display, but not change, the data of a part. Contrast with *part editor* .

part window -A window that displays an embedded part by itself, for easier viewing or editing. Any part that is embedded in another part can be opened up into its own part window. The part window is separate from and has a slightly different appearance than the *document window* that displays the entire document in which the part is embedded.

part-wrapper object -A private OpenDoc object that is used to reference a part.

partition -(1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

Paste -A choice in the **Edit** pull-down that a user selects to move the contents of the clipboard into a preselected location. See also *Copy* and *Cut* .

Paste As -A choice in the **Edit** pull-down that a user selects to move the contents of the clipboard into a preselected location by means of a dialog box allowing the user to specify the format of the data. See also *Copy* and *Cut* .

path -The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

PDD -Physical device driver.

peeking -An action taken by any thread in the process that owns the queue to examine queue elements without removing them.

pel -(1) The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point* , *pixel* , and *picture element* . (2) Picture element.

persistence -The quality of an entity such as a part, link, or object, that allows it to span separate document launches and transport to different computers. For example, a part unloaded to persistent storage is typically written to a hard disk.

persistent object -An object whose instance data and state are preserved between system shutdown and system startup.

persistent reference -A number, stored somewhere within a storage unit, that refers to another storage unit in the same document. Persistent references permit complex runtime object relationships to be stored externally, and later reconstructed.

physical device driver (PDD) -A system interface that handles hardware interrupts and supports a set of input and output functions.

pick -To select part of a displayed object using the pointer.

pickup -To add an object or set of objects to the pickup set.

pickup and drop -A drag operation that does not require the direct manipulation button to be pressed for the duration of the drag.

pickup set -The set of objects that have been picked up as part of a pickup and drop operation.

picture chain -See *segment chain* .

picture element -(1) Synonym for *pel* . (2) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) . (3) The area of the finest detail that can be reproduced effectively on the recording medium.

PID -Process identification.

pipe -(1) A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe* . (2) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (|).

pixel -(1) Synonym for *pel* . (2) Picture element.

platform -The operating system environment in which a program runs. For example, OpenDoc is implemented on the OS/2, Macintosh, and Windows platforms.

platform-normal coordinates -The native coordinate system for a particular platform. OpenDoc performs all layout and drawing in platform-normal coordinates; to convert from another coordinate system to platform-normal coordinates requires application of a *bias transform* .

plotter -An output unit that directly produces a hardcopy record of data on a removable medium, in the form of a two-dimensional graphic representation. (T)

PM -Presentation Manager.

pointer -(1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse* . The pointer is used to point at items that users can select. Contrast with *cursor* . (2) A data element that indicates the location of another data element. (T)

POINTERS\$ -Character-device name reserved for a pointer device (mouse screen support).

pointing device -In SAA Advanced Common User Access architecture, an instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pointings -Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse* .

polyfillet -A curve based on a sequence of lines. The curve is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet* .

polygon -One or more closed figures that can be drawn filled, outlined, or filled and outlined.

polyline -A sequence of adjoining lines.

polymorphism -The ability to have different implementations of the same method for two or more classes of objects.

pop -To retrieve an item from a last-in-first-out stack of items. Contrast with *push* .

pop-up menu -A menu that lists the actions that a user can perform on an object. The contents of the pop-up menu can vary depending on the context, or state, of the object.

pop-up window -(1) A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window. (2) In SAA Advanced Common User Access architecture, a movable window, fixed in size, in which a user provides information required by an application so that it can continue to process a user request.

port -(1) A connection between the CPU and main memory or a device (such as a terminal) for transferring data. (2) A socket on the back panel of a computer where you plug in a cable for connection to a network or a peripheral device.

port name -A unique identifier for a particular application within a computer. The port name contains a name string, a type string, and a script code. An application can specify any number of port names for a single port so long as each name is unique. See also *port* .

position code -A parameter (to a storage unit's Focus method) with which you specify the desired property or value to access.

preferences -The mechanism through which the user assigns a part editor to a given part kind.

preferred editor -The part editor that last edited a part, or for whom the part's data was just translated. If a part's preferred editor is not present, OpenDoc attempts to bind the part to the user's *default editor for kind* or *default editor for category* .

preferred kind -The part kind that a part specifies as its highest-fidelity, preferred format for editing. It is the part kind stored as the first value in the contents property of the part's storage unit, unless the storage unit also contains a property of type *kODPropPreferredKind* specifying another value as the preferred kind.

presentation -A particular style of display for a part's content-for example, an outline or expanded style for text, or a wire-frame or solid style for graphic objects. A part can have multiple presentations, each with its own rendering, layout, and user-interface behavior. Contrast with *view type* .

presentation drivers -Special purpose I/O routines that handle field device-independent I/O requests from the PM and its applications.

Presentation Manager (PM) -The interface of the OS/2 operating system that presents, in windows a graphics-based interface to applications and files installed and running under the OS/2 operating system.

presentation page -The coordinate space in which a picture is assembled for display.

presentation space (PS) -(1) Contains the device-independent definition of a picture. (2) The display space on a display device.

primary window -In SAA Common User Access architecture, the window in which the main interaction between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window* .

primitive -In computer graphics, one of several simple functions for drawing on the screen, including, for example, the rectangle, line, ellipse, polygon, and so on.

primitive attribute -A specifiable characteristic of a graphic primitive. See *graphics attributes* .

primitive object class -An object class defined in the *OSA Event Registry: Standard Suites* for OSA event objects that contain a single value; for example, the cBoolean, cLongInteger, and cAlias object classes are all primitive object classes. An OSA event object that belongs to a primitive object class has no properties and contains only one element of the value of the data.

Print Documents event -An OSA event that requests that an application print a list of documents; one of the four required OSA events.

print job -The result of sending a document or picture to be printed.

private header file -A SOM-generated file containing macros that provide access to instance variables and invoke superclass methods of a SOM class.

privilege level -A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment* .

procedure call -In programming languages, a language construct for invoking execution of a procedure.

process -An instance of an executing application and the resources it is using.

program -A sequence of instructions that a computer can interpret and execute.

program details -Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group -In the Presentation Manager, several programs that can be acted upon as a single entity.

program name -The full file specification of a program. Contrast with *program title* .

program title -The name of a program as it is listed in the *Program Manager* window. Contrast with *program name* .

promise -A specification of data to be transferred at a future time. If a data transfer involves a very large amount of data, the source part can choose to put out a promise instead of actually writing the data to a storage unit.

prompt -A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

Properties notebook -A control window that is used to display the properties for a part and to enable the user to change them.

property -(1) In the OpenDoc storage subsystem, a component of a storage unit. A property defines a kind of information (such as "name" or "contents") and contains one or more data streams, called *values* , that consist of information of that kind. Properties in a stored part are accessible without the assistance of a part editor. Contrast with *content property* and *Info property* . See also *display property*, *user property*, and *OSA event property* . (2) An OSA event object that defines some characteristic of another OSA event object, such as its font or point size, that can be uniquely identified by a constant. The definition of each object class in the *OSA Event Registry: Standard Suites* lists the constants and class IDs for properties of OSA event objects belonging to that object class. For example, the constants pName and pBounds identify the name and boundary properties of OSA event objects that belong to the object class cWindow. The pName property of a specific window is defined by an OSA event object of object class cProperty, such as the word "MyWindow," which defines the name of the window. An OSA event object can contain only one of each of its properties, whereas it can contain many elements of the same element class. See also *OSA event object* , *container* , *element classes* .

property ID -A four-character code, which can also be represented by a constant, that identifies an OSA property.

protect mode -A method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with *real mode* .

protocol -(1) A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I) (2) The programming interface through which a specific task or set of related tasks is performed. The drag-and-drop protocol, for example, is the set of calls that a part editor makes (and responds to) in order to support the dragging of items into or out of its content.

proxy -A special type of content element in a containing part, a proxy is the site of an embedded part. A proxy holds a frame that has a reference to an embedded part or linked part.

proxy content -Data, associated with a single embedded frame written to the Clipboard (or drag-and-drop object or link-source object), that the frame's original containing part wanted associated with the frame, such as a drop shadow or other visual adornment. Proxy content is absent if intrinsic content as well as an embedded frame was written.

pseudocode -An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. (A)

public header file -A SOM-generated file containing the client interface of a SOM class.

pull-down -(1) An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action

bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users. (2) In SAA Common User Access architecture, pertaining to a choice in an action bar pull-down.

purge -To free noncritical memory, usually by writing or releasing cached data. In low-memory situations, OpenDoc can ask a part editor or other objects to purge memory.

push -To add an item to a last-in-first-out stack of items. Contrast with *pop* .

push button -In SAA Advanced Common User Access architecture, a rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

putback -To remove an object or set of objects from the lazy drag set. This has the effect of undoing the pickup operation for those objects

putdown -To drop the objects in the lazy drag set on the target object.

Glossary - Q

queue -(1) A linked list of elements waiting to be processed in FIFO order. For example, a queue may be a list of print jobs waiting to be printed. (2) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

queued device context -A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context* .

Quit Application event -An OSA event that requests that an application perform the tasks-such as releasing memory, asking the user to save documents, and so on-associated with quitting; one of the four required OSA events. The Finder sends this event to an application immediately after sending it a Print Documents event or if the user chooses Restart or Shut Down from the Finder's Special menu.

Glossary - R

radio button -(1) A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from a list. Contrast with *check box* . (2) In SAA Advanced Common User Access architecture, a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which only one can be selected. The circle is partially filled when a choice is selected.

range descriptor record -A coerced AE record of type `typeRangeDescriptor` that identifies two OSA event objects marking the beginning and end of a range of elements. See also *boundary objects* .

RAS -Reliability, availability, and serviceability.

raster -(1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (T) (2) The coordinate grid that divides the display area of a display device. (A)

read-only file -A file that can be read from but not written to.

real mode -A method of program operation that does not limit or prevent access to any instructions or areas of storage. The operating system loads the entire program into storage and gives the program access to all system resources. Contrast with *protect mode* .

realize -To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recordable -A level of scripting support of a part. A recordable part allows the user to automatically convert user actions into scripts attached to the part. Contrast with *scriptable*, *customizable*, *tinkerable* .

recordable application -An application that uses OSA events to report user actions to the OSA Event Manager for recording purposes. When a user turns on recording (for example, by pressing the Record button in the Script Editor application), a scripting component translates the OSA events generated by the user's subsequent actions into statements in a scripting language and records them in a compiled script. See also *scriptable application* .

recordable event -Any OSA event that any recordable application sends to itself while recording is turned on for the local computer, with the exception of events that are sent with the `kAEDontRecord` flag set in the `sendMode` parameter of the `AESend` function.

recording process -Any process (for example, a script editor) that can turn OSA event recording on and off and receive and record recordable OSA events.

recursive routine -A routine that can call itself, or be called by another routine that was called by the recursive routine.

reentrant -The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference -A pointer to (or other representation of) an object, used to gain access to the object when needed.

reference count -The number of references to an object. Objects that are reference-counted, such as windows and parts, cannot be deleted from memory unless their reference counts are zero.

reference counted object -An object that maintains a reference count. All classes descended from ODRefCntObject are reference-counted.

reference phrase -(1) A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available. (2) In hypertext, text that is highlighted and preceded by a single-character input field used to signify the existence of a hypertext link.

reference phrase help -In SAA Common User Access architecture, highlighted words or phrases within help information that a user selects to get additional information.

refresh -To update a window, with changed information, to its current status.

region -A clipping boundary in device space.

register -A part of internal storage having a specified storage capacity and usually intended for a specific purpose. (T)

registry -A dictionary that lists executable code modules and associated data by which they can be selected. Examples of a registry are *part registry* and *scripting component registry*.

release -To delete a reference to an object. For a reference-counted object, releasing it decrements its reference count.

remote file system -A file-system driver that gains access to a remote system without a block device driver.

remove -To delete an object (such as a frame) permanently from its draft, as well as from memory. Contrast with *close*.

required OSA event -One of the four OSA events in the Required suite: Open Documents, Open Application, Print Documents, or Quit Application.

required parameter -An OSA event parameter that must be included in an OSA event. For example, a list of documents to open is a required parameter for the Open Documents event. Direct parameters are often required, and other additional parameters may be required. Optional parameters are never required.

resolve -To locate the OSA event object described by an object specifier record.

resource -The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file -A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore -To return a window to its original size or position following a sizing or moving action.

retained graphics -Graphic primitives that are remembered by the Presentation Manager interface after they have been drawn. Contrast with *nonretained graphics*.

return code -(1) A value returned to a program to indicate the results of an operation requested by that program. (2) A code used to influence the execution of succeeding instructions.(A)

reverse video -(1) A form of highlighting a character, field, or cursor by reversing the color of the character, field, or cursor with its background; for example, changing a red character on a black background to a black character on a red background. (2) In SAA Basic Common User Access architecture, a screen emphasis feature that interchanges the foreground and background colors of an item.

revert -To return a draft to the state it had just after its last save.

REXX Language -Restructured Extended Executor. A procedural language that provides batch language functions along with structured programming constructs such as loops; conditional testing and subroutines.

RGB -(1) Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light. (2) Pertaining to a color display that accepts signals representing red, green, and blue.

roman -Relating to a type style with upright characters.

root facet -The facet that displays the root frame in a document window.

root frame -The frame in which the root part of a document is displayed. The root frame shape is the same as the content area of the document window.

root part -The part that forms the base of a document and establishes its basic editing, embedding, and printing behavior. A document has only one root part, which can contain content elements and perhaps other, embedded parts. Any part can be a root part.

root segment -In a hierarchical database, the highest segment in the tree structure.

root storage unit -See *content storage unit* .

root window -See *document window* .

round-robin scheduling -A process that allows each thread to run for a specified amount of time.

run time -(1) Any instant at which the execution of a particular computer program takes place. (T) (2) The amount of time needed for the execution of a particular computer program. (T) (3) The time during which an instruction in an instruction register is decoded and performed. Synonym for *execution time* .

Glossary - S

SAA -Systems Application Architecture.

save -To write all the data of all parts of a document (draft) to persistent storage.

SBCS -Single-byte character set.

scheduler -A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

scope -The range of a cloning operation, limiting which objects are to be copied. Scope is expressed in terms of a frame object or its storage unit.

screen -In SAA Basic Common User Access architecture, the physical surface of a display device upon which information is shown to a user.

screen device context -A logical description of a data destination that is a particular window on the screen. See also *device context* .

SCREEN\$ -Character-device name reserved for the display screen.

script -A sequence of written instructions that, when executed by a script interpreter, are converted to semantic events that manipulate parts.

script application component -A component registered with the Component Manager at system startup. When a user opens a script application, the script application component loads the script and passes the resulting script ID to the appropriate scripting component for execution.

script data -A compiled script, script value, script context, or any other representation of a script in memory used internally by a scripting component. See also *compiled script* and *script value* .

script editor -An application that allows users to record, edit, save, and execute scripts; for example, the Script Editor application.

script file -A file in which a script is stored. A script file can be a compiled script file, a script application file, or a script text file.

script ID -A data structure of type OSAID-that is, a long integer-used by scripting components to keep track of script data.

script text file -Uncompiled statements in a scripting language saved by a script editor as a text file. A user must open a script text file in a script editor and successfully compile it before it will execute. See also *script editor* .

script value -An integer, a string, a Boolean value, a constant, or any other fixed data that a scripting component returns or uses in the course of executing a script.

scriptable -A level of scripting support of a part. A scriptable part is able to accept semantic events for its publicly published content objects and operations. Contrast with *customizable* , *tinkerable* and *recordable* .

scriptable application -An application that can respond as a server application to OSA events sent to it by scripting components. To be scriptable, an application must respond to the appropriate standard OSA events, and it must provide an 'aete' resource that describes the nature of that support. See also *OSA event user terminology resources* .

scripting -Writing and executing scripts to control the behavior of multiple applications.

scripting component -A component that responds appropriately to calls made to the standard scripting component routines. Most scripting

components implement scripting languages; for example, the Object REXX component implements the Object REXX scripting language.

scroll bar -In SAA Advanced Common User Access architecture, a part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently visible.

scrollable entry field -An entry field larger than the visible field.

scrollable selection field -A selection field that contains more choices than are visible.

scrolling -Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window -A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

sector -On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment -See *graphics segment* .

segment attributes -Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain -All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain* .

segment priority -The order in which segments are drawn.

segment store -An area in a normal graphics presentation space where retained graphics segments are stored.

select -(1) To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing. (2) In OpenDoc, to designate as the focus of subsequent editing operations. If the user selects an embedded part, that part's frame border takes on an appearance that designates it as selected. The embedded part's container is activated.

select button -The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor -In SAA Advanced Common User Access architecture, a visual indication that a user has selected a choice. It is represented by outlining the choice with a dotted box. See also *text cursor* .

selection field -(1) In SAA Advanced Common User Access architecture, a set of related choices. See also *entry field* . (2) In SAA Basic Common User Access architecture, an area of a panel that cannot be scrolled and contains a fixed number of choices.

selection focus -The location of editing activity. The part whose frame has the selection focus is the active part, and has the selection or insertion point. See also *keystroke focus* .

semantic event -A message sent to a part or one of its content elements. Semantic events pertain directly to the part's content model and can have meaning independent of the part's display context. For example, semantic events could direct a part to get, set, or delete data. Contrast with *user event* . See also *Open Scripting Architecture* .

semantic-event handler -A routine that executes in response to receiving a specific semantic event.

semantic interface -A set of OpenDoc objects that provides an interface to allow parts to receive messages (semantic events) from other parts, in the same document or in other documents.

semantics -The relationships between symbols and their meanings.

semaphore -An object used by applications for signalling purposes and for controlling access to serially reusable resources.

separator -In SAA Advanced Common User Access architecture, a line or color boundary that provides a visual distinction between two adjacent areas.

sequence number -A number that defines the position of a frame in its *frame group* .

serial dialog box -See *modal dialog box* .

serialization -The consecutive ordering of items.

serialize -To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR) -A logical resource or object that can be accessed by only one task at a time.

server application -An application that responds to OSA events requesting a service or information sent by client applications or scripting components (for example, by printing a list of files, checking the spelling of a list of words, or performing a numeric calculation). OSA event servers and clients can reside on the same local computer.

service -An OpenDoc component that, unlike a part editor, is not primarily concerned with editing and displaying parts. Instead, it provides a service to parts or documents, using the OpenDoc extension mechanism. Spelling checkers or database-access tools, for example, can

be implemented as services.

session -(1) A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files. (2) In the OS/2 operating system, one instance of a started program or command prompt. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses, such as computer memory, allocation of processor time, and windows on the screen. (3) A logical connection between two entities (such as an OS/2 program and a database server) that facilitates the transmission of information between the two entities. An application has the option to accept or reject a session request. Authentication of the requesting user may be required before a session can commence. See also *authentication* , *message block* , *port* .

session ID -A number that uniquely identifies a session.

Settings dialog box -A dialog box, accessible through the Part Info dialog box, that displays part-specific, custom Info properties.

settings extension -An OpenDoc extension class that you can use to implement a Properties notebook.

shadow -An object that refers to another object. A shadow is not a copy of another object, but is another representation of the object.

shadow box -The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shape -A description of a geometric area of a drawing canvas.

shared data -Data that is used by two or more programs.

shared memory -In the OS/2 operating system, a segment that can be used by more than one program.

shared resource -A facility used by multiple parts. Examples of shared resources are the menu focus, selection focus, keystroke focus, and serial ports. See also *arbitrator* .

shear -In computer graphics, the forward or backward slant of a graphics symbol or string of such symbols relative to a line perpendicular to the baseline of the symbol.

shell -(1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

shell plug-in -A shared library that modifies or extends the functions of the document shell.

shutdown -The process of ending operation of a system or a subsystem, following a defined procedure.

sibling -A frame or facet at the same level of embedding as another frame or facet within the same containing frame or facet. Sibling frames and facets are *z-ordered* to allow for overlapping.

sibling processes -Child processes that have the same parent process.

sibling windows -Child windows that have the same parent window.

signature -The aspect of a method defined by its return type and parameter list.

simple list -A list of like values; for example, a list of user names. Contrast with *mixed list* .

simple part -A part that cannot itself contain embedded parts. Contrast *container part* .

single-byte character set (SBCS) -A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set* .

slider box -In SAA Advanced Common User Access architecture: a part of the scroll bar that shows the position and size of the visible information in a window relative to the total amount of information available. Also known as *thumb mark* .

small icon view type -A view type in which a part is represented by a 16 by 16-pixel bitmap image. Other possible view types for displaying a part include large icon, thumbnail, and frame.

SOM -See *System Object Model* .

SOM object -An object or class created according to the system object model.

source application -The application that sends a particular OSA event to another application or to itself. Typically, an OSA event client sends an OSA event requesting a service from an OSA event server; in this case, the client is the source application for the OSA event. The OSA event server may return a different OSA event as a reply; in this case, the server is the source for the reply OSA event.

source content -The content at the source of a link. It is copied into the link and then into the *destination content* .

source data -Statements in a scripting language that constitute an uncompiled script.

source file -A file that contains source statements for items such as high-level language programs and data description specifications.

source frame -(1) An embedded frame whose part that has been opened up into its own *part window* . (2) The frame to which other *synchronized frames* are attached.

source part -(1) In data transfer, the part that provides the data that is transferred. (2) For a link, the part that contains the original information that is copied and displayed at the destination of the Contrast with *destination part* .

source statement -A statement written in a programming language.

special handler dispatch table -A table in shared memory that the OSA Event Manager uses to keep track of various specialized handlers.

specific dynamic-link module -A dynamic-link module created for the exclusive use of an application.

spin button -In SAA Advanced Common User Access architecture, a type of entry field that shows a scrollable ring of choices from which a user can select a choice. After the last choice is displayed, the first choice is displayed again. A user can also type a choice from the scrollable ring into the entry field without interacting with the spin button.

spline -A sequence of one or more Bézier curves.

split-frame view -A display technique for windows or frames, in which two or more facets of a frame display different scrolled portions of a part's content.

spooler -A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources from being intermixed.

stack -A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window -A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static canvas -A drawing canvas that cannot be changed after it has been rendered, such as a printer page. Contrast with *dynamic canvas* .

static control -The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage -(1) A read/write storage unit in which data is retained in the absence of control signals. (A) Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage* . (A)

stationery -A part that opens by copying itself and opening the copy into a window, leaving the original stationery part unchanged.

storage system -The OpenDoc mechanism for providing persistent storage for documents and parts. The storage system object must provide unique identifiers for parts as well as cross-document links. It stores parts as a set of standard properties plus type-specific content data.

storage unit -In the OpenDoc storage subsystem, an object that represents the basic unit of persistent storage. Each storage unit has a list of properties, and each property contains one or more data streams called values.

storage-unit cursor -A preset storage unit/ property/value designation, created to allow swift focusing on frequently accessed data.

storage-unit ID -A unique identifier of a storage unit within a draft.

storage-unit view -A storage unit prefocused on a given property and value. A storage-unit view provides thread-safe access to a storage unit.

strong persistent reference -A persistent reference that, when the storage unit containing the reference is cloned, causes the referenced storage unit to be copied also. Contrast with *weak persistent reference* .

style -See *window style* .

subclass -An object class that inherits properties, element classes, and OSA events from another object class-its superclass. A subclass can also include properties, element classes, or OSA events that are not inherited from its superclass. Every object class, with the exception of *cObject*, is a subclass of another object class. See also *object class* , and *superclass* .

subdirectory -In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

subframe -A frame that is both an embedded frame in, and a display frame of, a part. A part can create an embedded frame, make it a subframe of its own frame, and then display itself in that subframe.

subsystem -A broad subdivision of the interface and capabilities of OpenDoc, involving one or more protocols (for example, OpenDoc subsystems include shell, storage, drawing, user events, and semantic events).

suite -In the *OSA Event Registry: Standard Suites* , a group of definitions for OSA events, object classes, primitive object classes, descriptor types, and constants that are all used for a set of related activities. For example, the Text suite includes definitions of OSA events, object classes, and so on that are used for text processing.

superclass -A class from which another class (its *subclass*) is derived. Also called ancestor, base class, or parent class. It is the object class from which a subclass inherits properties, elements, and OSA events. See also *inheritance* , *object class* , *subclass* .

swapping -(1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (I) (A) (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch -(1) In SAA usage, to move the cursor from one point of interest to another; for example, to move from one screen or window to another or from a place within a displayed image to another place on the same displayed image. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list -See *Task List* .

symbolic identifier -A text string that equates to an integer value in an include file, which is used to identify a programming object.

symbols -In Information Presentation Facility, a document element used to produce characters that cannot be entered from the keyboard.

synchronized frames -Separate frames that display the same representation of the same part, and should therefore be updated together. In general, if an embedded part has two or more editable display frames of the same presentation, those frames (and all their embedded frames) should be synchronized.

synchronous -Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T)
See also *asynchronous* .

synthetic command ID -A command ID created by OpenDoc for a menu command that had not previously been registered with the menu bar object.

system coercion dispatch table -See *coercion handler dispatch table* .

System Menu -In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system object accessor dispatch table -See *object accessor dispatch table* .

System Object Model (SOM) -A mechanism for language-neutral, object-oriented programming.

system OSA event dispatch table -See *OSA event dispatch table* .

system queue -The master queue for all pointer device or keyboard events.

system result handler -A result handler that is available to all applications that use the system. Contrast with *application result handler* .

system-defined messages -Messages that control the operations of applications and provides input an other information for applications to process.

Systems Application Architecture (SAA) -A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

Glossary - T

table tags -In Information Presentation Facility, a document element that formats text in an arrangement of rows and columns.

tag -(1) One or more characters attached to a set of data that contain information about the set, including its identification. (I) (A) (2) In Generalized Markup Language markup, a name for a type of document or document element that is entered in the source document to identify it.

target address -An application signature, a process serial number, a session ID, a target ID record, or some other application-defined type that identifies the target of an OSA event.

target application -The application addressed to receive an OSA event. Typically, an OSA event client sends an OSA event requesting a service from a server application; in this case, the server is the target application of the OSA event. The server application may return a different OSA event as a reply; in this case, the client is the target of the reply OSA event.

target object -An object to which the user is transferring information.

Task List -In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

terminate-and-stay-resident (TSR) -Pertaining to an application that modifies an operating system interrupt vector to point to its own location (known as hooking an interrupt).

terminology resource -A resource (of type 'aete') that is required for scriptability.

text -Characters or symbols.

text cursor -A symbol displayed in an entry field that indicates where typed input will appear.

text window -Also known as the VIO window.

text-windowed application -The environment in which the operating system performs advanced-video input and output operations.

thread -A unit of execution within a process. It uses the resources of the process.

thread-safe -Said of an activity, or access to data, that can be safely undertaken in a multitasking environment.

thumb mark -The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

thumbnail view type -A view type in which a part is represented by a large (64-by-64 pixels) bitmap image that is typically a miniature representation of the layout of the part content. Other possible view types for displaying a part include large icon, small icon, and frame.

thunk -Term used to describe the process of address conversion, stack and structure realignment, etc., necessary when passing control between 16-bit and 32-bit modules.

tilde -A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time-critical process -A process that must be performed within a specified time after an event has occurred.

time slice -(1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

timer -A facility provided under the Presentation Manager, whereby Presentation Manager will dispatch a message of class WM_TIMER to a particular window at specified intervals. This capability may be used by an application to perform a specific processing task at predetermined intervals, without the necessity for the application to explicitly keep track of the passage of time.

timer tick -See *clock tick*.

tinkerable -A level of scripting support of a part. A tinkerable part allows the user to customize it, changing its behavior during virtually any user action. Contrast with *scriptable* and *recordable*.

title bar -In SAA Advanced Common User Access architecture, the area at the top of each window that contains the window title and system menu icon. When appropriate, it also contains the minimize, maximize, and restore icons. Contrast with *panel title*.

TLB -Translation lookaside buffer.

token -A short, codified representation of a string. The session object creates tokens for ISO strings. In OSA events for OpenDoc, a special descriptor structure that a part uses to identify one or more content objects within itself.

token disposal function -An object callback function that disposes of a token.

transaction -(1) An exchange between a workstation and another device that accomplishes a particular action or result. (2) In OpenDoc, a sequence of OSA events sent back and forth between the client and server applications, beginning with the client's initial request for a service. All OSA events that are part of one transaction must have the same transaction ID.

transform -(1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

translation -The conversion of one type of data to another type of data. Specifically, the conversion of data of one part kind to data of another part kind. Note that translation can involve loss of *fidelity*.

Translation lookaside buffer (TLB) -A hardware-based address caching mechanism for paging information.

Tree -In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate -(1) To terminate a computational process in accordance with some rule (A) (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

Glossary - U

undo -To rescind a command, negating its results. OpenDoc provides the ability to undo events by utilizing a command history.

unnamed pipe -A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

unordered list -In Information Presentation Facility, a vertical arrangement of items in a list, with each item in the list preceded by a special character or bullet.

update region -A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect, and therefore are in need of repainting.

update ID -(1) In OpenDoc, a number used to identify a particular instance of Clipboard contents. (2) A number used to identify a particular instance of link source data.

used shape -A shape that describes the portion of a frame that a part actually uses for drawing; that is, the part of the frame that the containing part should not draw over.

user event -A message, sent to a part by the dispatcher, that pertains only to the state of the part's graphical user interface, not directly to its contents. User events include mouse clicks and keystrokes, and they deliver information about, among other things, window locations and scroll bar positions. Contrast with *semantic event*.

user interface -Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

user-interface part -A part without content elements, representing a unit of a document's user interface. Buttons and dialog boxes, for example, can be user-interface parts.

user property -One of a set of user-accessible characteristics of a part or its frame. The user can modify some user properties, such as the name of a part; the user cannot modify some other user properties, such as part category. Each user property defined by OpenDoc is stored as a distinct property in the storage unit of the part or its frame.

utility program -(1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

Glossary - V

validate -To mark a portion of a canvas (or facet, or frame) as no longer in need of redrawing. Contrast with *invalidate*.

value -In the OpenDoc storage subsystem, a data stream associated with a property in a storage unit. Each property has a set of values, and there can be only one value of a given data type for each property.

value set control -A visual component that enables a user to select one choice from a group of mutually exclusive choices.

vector font -A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font*. Contrast with *image font*.

VGA -Video graphics array.

view -A way of looking at an object's information.

view type -The basic visual representation of a part. Supported view types include frame, icon, small icon, and thumbnail.

viewer -See *part viewer*.

viewing pipeline -The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window -A clipping boundary that defines the visible part of model space.

VIO -Video Input/Output.

virtual memory (VM) -Synonymous with *virtual storage* .

virtual storage -(1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (I) (A) (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. (3) Synonymous with *virtual memory* .

visible region -A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume -(1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (I) (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

Glossary - W

weak persistent reference -A persistent reference that, when the storage unit containing the reference is cloned, is ignored; the referenced storage unit is not copied. Contrast with *strong persistent reference* .

whose descriptor record -A coerced AE record of descriptor type typeWhoseDescriptor. The OSA Event Manager creates whose descriptor records when it resolves object specifier records that specify formTest.

whose range descriptor record -A coerced AE record of type typeWhoseRange. Under certain conditions, the OSA Event Manager coerces a range descriptor record to a whose range descriptor record when it resolves object specifier records that specify formTest.

wildcard character -Synonymous with *global file-name character* .

window -(1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen. (3) A division of a screen in which one of several programs being executed concurrently can display information.

window canvas -The canvas attached to the root facet of a window. Every window has a window canvas.

window class -The grouping of windows whose processing needs conform to the services provided by one window procedure.

window-content transform -The composite transform that converts from a part's content coordinates to its window coordinates.

window coordinate space -The coordinate space of the window in which a part's content is drawn. It may or may not be equal to the canvas coordinate space.

window coordinates -A set of coordinates by which a window position or size is defined; measured in device units, or *pe/s* .

window-frame transform -The composite transform that converts from a part's frame coordinates to its window coordinates.

window handle -Unique identifier of a window, generated by Presentation Manager when the window is created, and used by applications to direct messages to the window.

window procedure -Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle -The means by which the size and position of a window is described in relation to the desktop window.

window resource -A read-only data segment stored in the .EXE file of an application or the .DLL file of a dynamic link library.

window state -An object that lists the set of windows that are open at a given time. Part editors can alter the window state, and the window state can be persistently stored.

window style -The set of properties that influence how events related to a particular window will be processed.

window title -In SAA Advanced Common User Access architecture, the area in the title bar that contains the name of the application and the OS/2 operating system file name, if applicable.

Workplace Shell -The OS/2 object-oriented, graphical user interface.

workstation -(1) A display screen together with attachments such as a keyboard, a local copy device, or a tablet. (2) One or more programmable or nonprogrammable devices that allow a user to do work.

world-coordinate space -Coordinate space in which graphics are defined before transformations are applied.

world coordinates -A device-independent Cartesian coordinate system used by the application program for specifying graphical input and output. (I) (A)

wrapper -An object (or class) that exists to provide an object-oriented interface to a non-object- oriented or system-specific structure. The OpenDoc class ODWindow, for example, is a wrapper for a system-specific window structure.

WYSIWYG -What-You-See-Is-What-You-Get. A capability of a text editor to continually display pages exactly as they will be printed.

Glossary - X

There are no glossary terms for this starting letter.

Glossary - Y

There are no glossary terms for this starting letter.

Glossary - Z

z-order -The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

z-ordering -The front-to-back ordering of sibling frames used to determine clipping and event handling when frames overlap.

zooming -The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)
